195 PTAS.
(IVA Incluido)

106

MI COMPUTER

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR



Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen IX-Fascículo 106

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22 - Télex: 93392 EPPA

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 120 fascículos de aparición semanal, encuadernables en diez volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-181-X (tomo 9)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 288601
Impreso en España-Printed in Spain-Enero 1986

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (120 fascículos más las tapas, guardas y transferibles para la confección de los 10 volúmenes) son las siguientes:

a) Un pago único anticipado de 27 105 ptas. o bien 10 pagos trimestrales anticipados y consecutivos de 2 711 ptas. (sin gastos de envío).

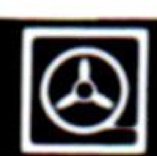
b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

No se efectúan envíos contra reembolso.







Cortesía de Mecca Bookmakers

Apuestas seguras

Atractivas visualizaciones
Las visualizaciones de información por ordenador mejoradas constituyen apenas una de las muchas aplicaciones a las que las cadenas de apuestas británicas más importantes están destinando la nueva tecnología. Entre otros usos se incluye la monitorización de la actividad de los clientes, la productividad del personal y el cálculo de las posibilidades de opciones de apuestas especiales

Los programas escritos para pronosticar ganadores en las carreras de caballos deben considerar factores tales como las estadísticas y la ley de las probabilidades

Los corredores de apuestas casi podrían considerarse como unos intermediarios que proporcionan un servicio, ofreciendo a los apostadores opiniones diferentes sobre los méritos de los caballos participantes en una carrera dada para que respalden sus apreciaciones con dinero contante y sonante. Las posibilidades que ofrecen los corredores de apuestas reflejan el peso del dinero apostado: cuantas más personas respalden un caballo, más dinero hay para él y, en consecuencia, "menor" es su precio. Además, los corredores intentan calibrar las posibilidades de modo que, en general, estén seguros de quedarse con un porcentaje del total, cualquiera que sea el caballo ganador. De ese modo, en realidad los apostadores pagan los servicios del corredor.

En Gran Bretaña, el mercado de apuestas en realidad cobra fuerza *in situ* en los diez minutos, más o menos, de frenética actividad que preceden a la carrera. Las sesiones de apuestas se transmiten a las agencias de todo el país, de modo que millones de apostadores que no están presentes en el hipódromo puedan hacer su apuesta.

A pesar de los obstáculos para la introducción de la nueva tecnología en la pista, algunos corredores exteriores a las pistas (en particular las grandes cadenas) están confiando cada vez más en los ordenadores para ganar en eficacia y rentabilidad. La firma Ladbrokes, por ejemplo, utiliza en gran medida ordenadores en su sección de apuestas a crédito. Todas las apuestas de los clientes a crédito se registran en micros Cromemco conectados a un disco rígido. Al terminar el día, los datos se transfieren a cinta y se envían a un ordenador central IBM al cual ya se le han proporcionado los resultados de todas las carreras de la jornada. Todas las apuestas, ya sean ganadoras directas, *each-way* o las diversas combinaciones y acumuladores, se determinan, entonces, de forma automática. Las cuentas de los clientes se actualizan semanalmente, y se libran cheques o se realizan requerimientos de pago. Además, el ordenador central se utiliza para las tareas estándares de procesamiento de datos propias de cualquier gran negocio.

En la medida en que las firmas más importantes llegan a considerarse a sí mismas no tanto como corredores sino más bien como una suerte de industria del ocio, existe una creciente tendencia a la utilización de ordenadores. Mecca Bookmakers, por ejemplo, confía en que la reducción del costo de la potencia de los ordenadores allanará el camino hacia un análisis de sus negocios mucho más sofisticado. Se podrían utilizar terminales situados en sucursales seleccionadas para realizar una profunda investigación de las preferencias de los apostadores, haciendo posible la identificación, por ejemplo, de la clase más popular de apuestas, de las facilidades que a la gente le gustaría que se les proporcionara, o del tipo de administración adecuado para incrementar el negocio.

Mecca piensa que la investigación de mercados

asistida por ordenador irá adquiriendo mayor importancia cuando la nueva legislación británica permita, finalmente, que las agencias de apuestas se conviertan en lugares más agradables y acogedores para el público. La empresa ya está utilizando su ordenador central para mejorar las visualizaciones gráficas de información sobre carreras de caballos y otros eventos deportivos, transmitiéndolas en forma espectacular y colorida a oficinas seleccionadas de la cadena. Además, los ordenadores están ayudando a monitorizar el rendimiento y la rentabilidad de las agencias individuales y del grupo como un todo, asegurando que la dirección sepa claramente si está alcanzando o no sus objetivos.

En la actualidad, la mayoría de los corredores exteriores a las pistas ofrecen la posibilidad de predecir los dos o tres primeros corredores por el orden correcto. Estas apuestas se conocen, respectivamente, como el Computer Straight Forecast (CSF: pronóstico directo por ordenador). En ambos casos, el pago se calcula por ordenador de acuerdo con complejas fórmulas previamente establecidas. Cuando se iniciaron los pronósticos de carreras hípicas, algunos corredores se limitaban a multiplicar las posibilidades con el fin de calcular las apuestas ganadoras. No obstante, enseguida se comprendió que su dependencia de los caprichos de las posibilidades de las carreras los dejaba con márgenes inadecuados en el campo de las apuestas pronosticadas, en especial cuando, como sucede frecuentemente, los caballos favoritos terminan primero y segundo.

La fórmula de pago

En Gran Bretaña se ha desarrollado una fórmula para determinar el pago de cada combinación de posibilidades de modo que a los corredores se les garantizaba un margen del 20 %. En 1977 la fórmula se informatizó y nació el CSF, seguido en 1981 por el Tricast. Se introducen modificaciones al programa original (escrito en BASIC) en respuesta a los cambios que se producen en los patrones de apuestas o a los rendimientos del precio de partida.

La alteración más sonada tuvo lugar tras lo que se conoció como el "asunto de Little Owl". En enero de 1982, una carrera de vallas de tres caballos involucraba a un gran favorito (Little Owl), con 11-4 *on* (es decir, 4-11), un segundo favorito con 5-2 y un *rank outsider* con 66-1. Por algún motivo, Little Owl fue frenado recién comenzada la carrera, dejando que el segundo favorito (5-2) llegara a la meta delante del *outsider*. Para algunos corredores de apuestas constituyó un auténtico descalabro. Otros insinuaron veladamente que habían sido víctimas de una maniobra fraudulenta. La adición de una corrección (el "factor armónico") en el programa original aseguró que en competidores que produjeran resultados sorpresa las posibilidades de los *rank outsiders* se redujeran, mientras que las de caballos que gozaran de mayor aceptación se ampliaran ligeramente.

En consecuencia, el ordenador proporciona a los grandes corredores de apuestas exteriores a las pistas una poderosa herramienta para analizar su negocio, identificar áreas de vulnerabilidad y proteger y mejorar sus márgenes, de suma importancia.

Por cuanto concierne al apostador, hay un hecho innegable. Ningún programa diseñado para usar



Desde el hipódromo

Los corredores de apuestas de los hipódromos británicos han desarrollado un complejo sistema de interacción y de comunicación diseñado para satisfacer las demandas del mercado y ponerse a cubierto de las contingencias financieras. Los corredores se dividen en dos categorías: primero están los de mayores vuelos (o *corredores de barandilla*, así llamados debido a la posición que ocupan, junto a las barandillas que separan del público el recinto para socios), quienes sólo aceptan apuestas de clientes acreditados y originan el mercado. Luego están los de la zona del público (*Tattersalls*), quienes aceptan apuestas en efectivo de los espectadores.

En el lapso de unos pocos minutos agitados, antes de la carrera, el mercado de apuestas se convierte en un centro de gran actividad, durante el cual grandes sumas de dinero cambian de mano en miles de transacciones separadas. Durante este período, los corredores se comunican entre sí a través de un *hombre tictac*, quien utilizando sus manos transmite mensajes entre las partes mediante un complicado código de señales. Una de las funciones del *hombre tictac* es facilitar el proceso de "marcar" apuestas. Si un corredor toma conciencia de que se ha puesto a sí mismo en posición de posibles pérdidas con un determinado caballo, "cargará la responsabilidad", colocando, a través del *hombre tictac*, una apuesta sobre ese mismo caballo con otro corredor. Así, si el caballo gana, podrá recuperar parte de sus pérdidas.

Parece poco probable que el ordenador llegue alguna vez a participar en prácticas tan tradicionales, por la sencilla razón de que todo ocurre muy rápido. Comparativamente, sería fácil escribir un programa que simulara las actividades del corredor de apuestas, pero la entrada de los datos sería excesivamente lenta como para hacer frente al torbellino final de apuestas que tiene lugar para cada carrera. Parece ser, entonces, que el ordenador ha hallado en el *hombre tictac* la horma de su zapato

con un micro personal va a convertir al usuario en millonario de la noche a la mañana ni a sacar del negocio al corredor de apuestas. Al saber esto, la mayoría de los posibles apostadores por ordenador podrían sentirse desanimados; pero el micro puede ayudar al apostador juicioso en numerosos sentidos que merecen ser examinados.

Si bien es cierto que por lo general los apostadores deben perder (el margen del corredor y el impuesto sobre las apuestas velan por ello), sería erróneo suponer que todos ellos pierden. Los apostadores más perspicaces (aquellos que poseen una habilidad notoria para leer el formulario y sopesar las posibilidades) ganan, y sus ganancias provienen de quienes son menos capaces. El apostador juicioso dedica considerable tiempo al análisis de los rendimientos de los caballos con el objeto de establecer una imagen clara de sus méritos relativos.

Si el campo se puede reducir a dos o tres con posibilidades evidentes, entonces vale la pena considerar la carrera. Descubrir un caballo con unas *chances* sobresalientes es difícil, pero ocasionalmente viable. Luego se deben tener en cuenta otros factores, tales como la idoneidad del caballo, las condiciones de la carrera (distancia, *going*, jockey, etcétera) y, lo que es vital, la apuesta. Lo que le den por su dinero, por supuesto, es crucial: hallar un caballo con una *chance* obvia con buenas posibilidades. Esto, sin embargo, no es fácil, puesto que, al fin y al cabo, si las *chances* de un caballo son absolutamente evidentes, aun el apostador más incompetente aprovechará la oportunidad, lo que sin duda redundará en que el caballo salga con posibilidades restringidas.

La tarea del apostador de éxito consiste, por consiguiente, en hallar un caballo con *chances* abrumadoras que no atraiga un gran flujo de dinero y que, en consecuencia, es probable que salga con mayores posibilidades que las que debiera. Identificar tales oportunidades de apuesta lleva su tiempo y sólo se consigue raramente, de modo que el principio de una selectividad rigurosa es vital para la estrategia del apostador exitoso.

Simplemente, no hay tiempo para evaluar el formulario en cada una de las seis carreras, como mínimo, de cada reunión hípica. Dado que los apostadores de agencias de apuestas tienen acceso entre dos y cinco reuniones hípicas por tarde, apenas sorprende que la inmensa mayoría opere sobre una consideración del formulario que es poco mejor que una adivinanza lisa y llana. Por supuesto, en ocasiones este enfoque funciona, pero la tendencia general se orienta inexorablemente a la acumulación de pérdidas.

Es aquí donde entran en escena los programas para el apostador. La virtud de estos programas es que imponen al apostador un nivel de selectividad y, por tanto, impiden el enfoque indisciplinado, que es la perdición de tanta gente. Los programas hacen esto debido a lo que algunos considerarían uno de sus principales inconvenientes: la lentitud en la entrada de datos, que puede requerir alrededor de media hora para cada carrera hasta haber entrado todas las variables relevantes. Claro está que en principio el apostador sólo debe considerar aquellas carreras que ofrezcan las mayores posibilidades. Una estrategia recomendable es apostar generalmente en carreras de diez corredores o menos, con lo que se reduce el tiempo de entrada y, a la vez, se aumentan las posibilidades de ganar.

Es improbable, sin embargo, que el apostador por ordenador pueda hacer peligrar los intereses de los corredores de apuestas. El hecho es que tales programas sólo resultan atractivos al apostador juicioso y selectivo, desde cuyo punto de vista probablemente sea mejor que este estado de cosas continúe. Cualquier cambio radical en los patrones de apuestas será rápidamente identificado por la comunidad de corredores de apuestas, y las fuerzas del mercado inevitablemente se combinarán para asegurar que, suceda lo que suceda, el "hombre del talego" se gane su porcentaje al final del día.

Posibilidades razonables
Esta tabla es una muestra típica de las que publica frecuentemente el periódico británico *The Sporting Life*. Ofrece un análisis del rendimiento de un entrenador determinado (en este caso, David Elsworth) durante el año que se indica. Las cifras se desglosan bajo varios subtítulos que pueden interesar al apostador. Sin el ordenador, tales análisis serían imposibles desde el punto de vista del consumo de tiempo, y sólo se han convertido en realidad desde hace pocos años

1984 Flat statistics

	No. of Horses	Races Run	1st	2nd	3rd	Unpl	Per cent	£1 Level Stake
2-y-o	15	44	2	4	8	30	4.5	- 14.00
3-y-o	25	119	16	7	13	83	13.4	- 19.90
4-y-o+	14	51	6	5	4	36	11.8	+ 3.33
Totals	54	214	24	16	25	149	11.2	- 30.57

	Mar/Apr	May	Jun	Jul	Aug	Sep	Oct/Nov
2-y-o	0-0	0-3	1-9	0-4	0-7	0-10	1-11
3-y-o	0-6	4-24	1-19	4-21	1-17	2-13	4-19
4-y-o+	2-6	0-7	1-9	0-5	2-9	1-7	0-8
Totals	2-12	4-34	3-37	4-30	3-33	3-30	5-38

Non-Handicaps	W-R	£1 Level Stake
2-y-o	2-37	- 7.00
3-y-o	8-77	- 22.65
4-y-o+	0-4	- 4.00
Selling	2-8	- 0.17
Apprentice	1-1	+ 7.00
Amateur/Ladies	0-0	0.00
Totals	13-127	- 26.82

Handicaps	W-R	£1 Level Stake
2-y-o	0-3	- 3.00
3-y-o	5-36	- 10.25
4-y-o+	4-40	+ 2.00
Selling	1-3	+ 1.50
Apprentice	1-2	+ 9.00
Amateur/Ladies	0-3	- 3.00
Totals	11-87	- 3.75

Course Grade	W-R	%	£1 Level Stake
Group 1	7-100	7.0	- 35.50
Group ?	9-56	16.1	- 0.65
Group	4-43	9.3	- 8.50
Group 4	4-15	26.7	+ 14.08

First Time Out	W-R	%	£1 Level Stake
2-y-o	1-15	6.7	- 2.00
3-y-o	0-24	-	- 24.00
4-y-o+	1-12	8.3	- 4.00
Totals	2-51	3.9	- 30.00

Jockeys Riding	W-R	£1 Level Stake
B Rouse	11-94	- 23.92
S Cauthen	6-20	+ 33.25
D Brown	2-18	+ 1.00
R Fox	2-20	+ 10.00
P Cook	1-4	- 1.90
L Piggott	1-6	- 1.50
A McGlone	1-24	- 19.50

Pat Eddery	0-3	J Reid	0-2	R P Elliott	0-2
G Duffield	0-2	C Asmussen	0-1	R Cochrane	0-1
J H Brown	0-1	A Mackay	0-1	E Hide	0-1
Paul Eddery	0-1	Miss S Lawrence	0-1	J Matthias	0-1
J Mercer	0-1	B Procter	0-1	B Jago	0-1
W Carson	0-1	Mr M Bosley	0-1	Mr P Thomas	0-1
D Gibson	0-1	R Lines	0-1	M L Thomas	0-1
T Ives	0-1	T Quinn	0-1		

Course Record	Non-handicaps 2-y-o	Non-handicaps 3-y-o+	Handicaps 2-y-o	Handicaps 3-y-o+	Total W-R	£1 Level Stake
Brighton	0-2	3-10	0-0	1-1	4-13	+ 14.25
Folkestone	1-4	1-4	0-0	1-5	3-13	+ 11.58
Windsor	0-7	2-9	0-0	1-11	3-27	+ 0.50
Salisbury	1-6	0-10	0-1	2-14	3-31	- 7.50
Ascot	0-4	1-3	0-1	1-12	2-20	+ 7.00
Redcar	0-0	1-1	0-0	0-1	1-2	+ 0.10
Warwick	0-0	0-0	0-0	1-2	1-2	+ 2.50
York	0-0	0-0	0-0	1-2	1-2	+ 7.00
Bath	0-2	0-4	0-0	1-3	1-9	- 2.00
Lingfield	0-4	1-5	0-0	0-1	1-10	- 7.50
Kempton	0-2	0-3	0-0	1-6	1-11	+ 4.00
Newmarket	0-2	0-4	0-0	1-6	1-12	- 7.50
Goodwood	0-1	1-8	0-0	0-5	1-14	- 10.50
Newbury	0-1	1-11	0-1	0-2	1-15	- 9.50

Sandown	0-10	Doncaster	0-6	Epsom	0-6
Chepstow	0-3	Haydock	0-3	Leicester	0-3
Ayr	0-1	Nottingham	0-1		

Hemos llegado a un punto en el que estamos en condiciones de añadir elementos de la "trama". Examinaremos las últimas decisiones que es necesario adoptar en nuestro programa del manipulador de personajes y esbozaremos las estructuras arborescentes implicadas.

La trama se cierra

Hasta ahora hemos proporcionado a nuestros personajes los medios para que manipulen objetos. Lo que necesitamos ahora es añadir las rutinas restantes, que esbozamos en el capítulo anterior. Es en este punto donde se hacen evidentes las ventajas de adoptar un diseño modular para nuestro programa.

El diagrama de flujo de la ilustración representa el proceso completo de toma de decisiones que lleva a cabo el manipulador de personajes. Puede ver que incorpora tanto el árbol de manipulación de objetos, que ya hemos visto en acción, como módulos para manejar la trama, la "conciencia de objetos" y la interacción con otros personajes.

Los rudimentos de la trama ya los hemos analizado: en algún momento durante el curso del juego un infortunado personaje comerá una empanada de carne del Dog and Bucket y morirá por intoxicación. El juego terminará cuando otro personaje haya reunido la información necesaria para demostrar la culpabilidad del barman Fred, cuya costumbre de elaborar las empanadas con alimento para gatos ha dado lugar a este desventurado giro de los acontecimientos. Para resolver el misterio, un personaje debe hallar una lata de alimento para gatos, ver a la víctima y sumar dos más dos.

A los fines de la programación, la información necesaria se almacena en cuatro banderas principales. La primera de ellas ya la hemos visto: la variable numérica g. Ésta se actualiza en la línea 5220 y almacena temporalmente el número del personaje que está comiendo la empanada de carne para que las rutinas de la trama lo procesen. Otras dos banderas registran el fallecimiento de un personaje y si algún personaje ha detectado o no a la víctima. La cuarta bandera asume la forma de una matriz DIMensionada para la cantidad de personajes, con cada elemento inicializado en cero. Cuando un personaje descubre a la víctima, el elemento correspondiente se establece en 255. En el listado completo, que ofreceremos en el próximo capítulo, podremos ver todo esto en acción.

El árbol de la trama comprueba primero si el personaje ya ha muerto y, de ser así, la bandera de la "muerte" se establecerá en el número del personaje. Dado que el programa sólo admite una víctima, en este punto la rutina de la trama retornará sin emprender ninguna otra acción.

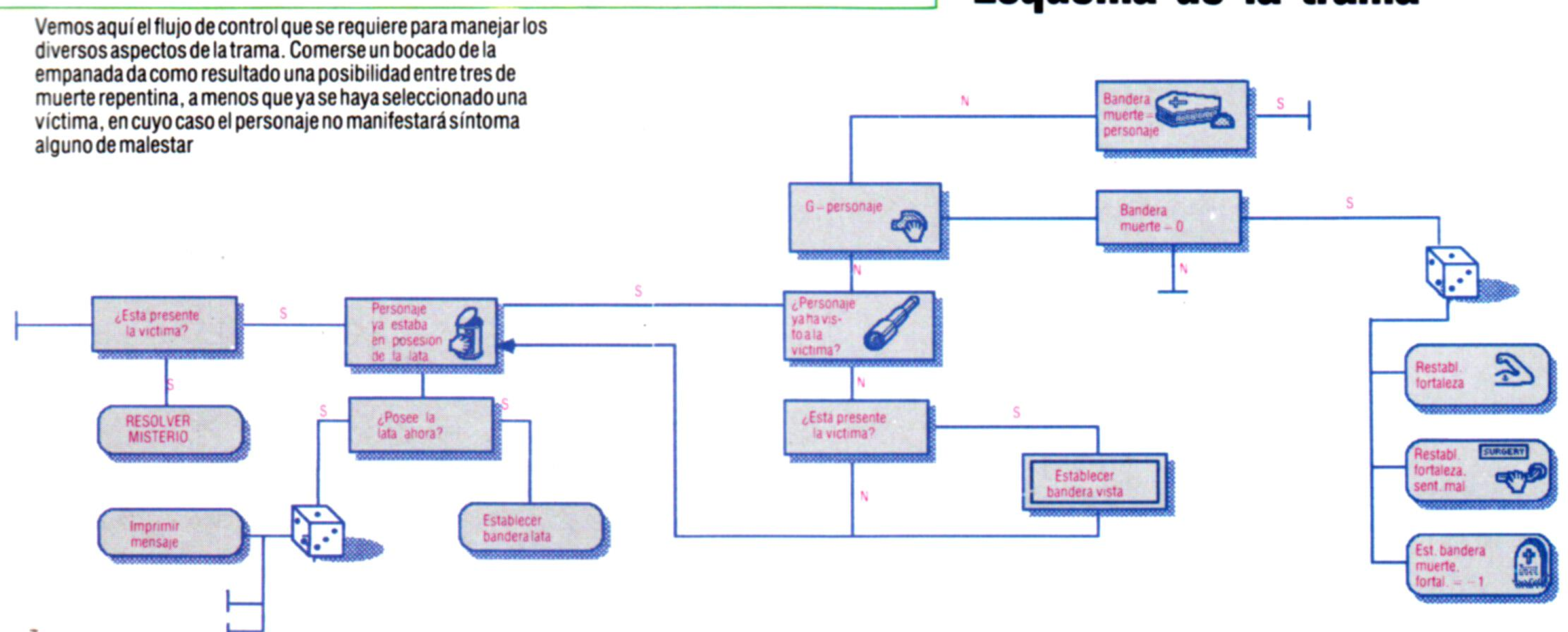
Sin embargo, si la bandera de la "muerte" no está establecida, la rutina prosigue comprobando si el personaje ha comido o no un bocado de la empanada. Si el valor de g concuerda con el número de personaje actual, entonces comprobamos si se ha producido alguna muerte (en cuyo caso la bandera de la "muerte"

Manejando el reparto

Llamar al manipulador de personajes
¿Mover al personaje?
Actualización de la trama
Bifurc aleatoria
Bifurc aleatoria
Manipulación de objetos
Interacción con personajes
Conciencia de objetos
Actividad general
Retorno al programa principal

Nuestro diagrama de flujo muestra las diferentes acciones que se emprenden cuando el programa principal pasa el control al módulo manipulador de personajes. Observe que el mover a un personaje impide cualquier otra acción, al objeto de evitar dificultades que podrían ocurrir si un personaje hubiera de interactuar con su entorno y cambiar de escenario durante la misma pasada de la rutina manipuladora

Esquema de la trama



Vemos aquí el flujo de control que se requiere para manejar los diversos aspectos de la trama. Comerse un bocado de la empanada da como resultado una posibilidad entre tres de muerte repentina, a menos que ya se haya seleccionado una víctima, en cuyo caso el personaje no manifestará síntoma alguno de malestar

Ian McKinnell

Conciencia de objetos

¡Ah!... ¡Esto es mejor!
S
¿Posee sus propios objetos?
N
Humor = humor − 1
Buscar bebida
¿Quién tiene mi bebida?

Aquí añadimos a las rutinas que proporcionaba el árbol de manipulación de objetos algunas otras orientadas a éstos. En especial, el humor del personaje se reduce en uno si no se halla en posesión de su propia bebida

Interacción de personajes

Nuestros personajes necesitan tener cierta conciencia del humor y las acciones de cada uno de los demás. Este árbol les proporciona algunas oportunidades de expresarse por sí mismos

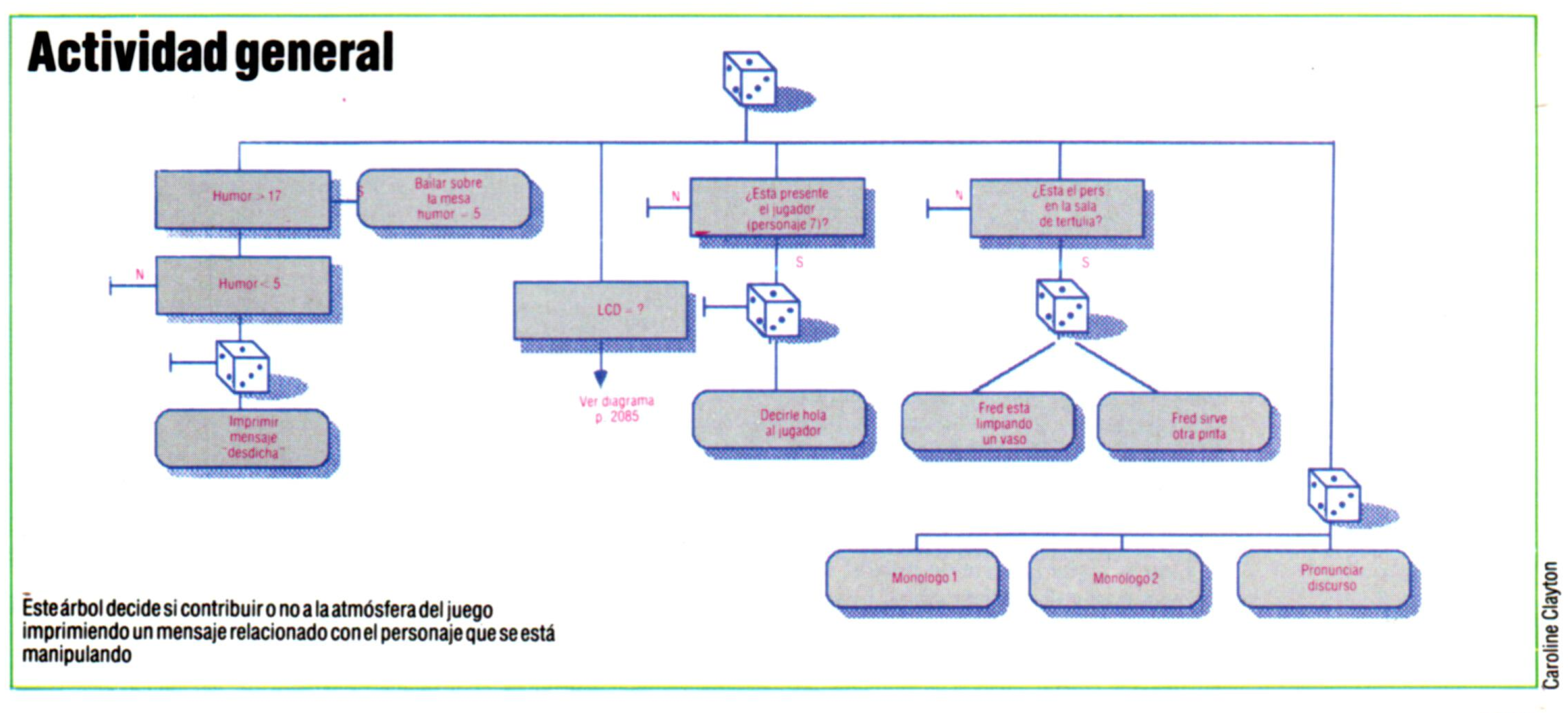
Seleccionar un personaje
Ninguno presente
Fortaleza < 0
S
Bandera muerte = personaje
S
N
Imprimir mensaje de inconsciente
N
¿El personaje tiene tu bebida?
S
Imprimir mensaje de indignación
N
¿Humor del otro personaje > 15?
S
"¡Se te ve muy alegre!"
N
¿Humor del personaje manipulado > 15?
S
Los personajes bailan juntos
N
¿El personaje acaba de entrar?
S
Decir hola, etc.

será mayor que cero). Si nadie ha fallecido, el programa se bifurca al azar a una de tres conclusiones. En las dos primeras, el personaje no presentará ningún síntoma de enfermedad o bien sufrirá un repentino dolor de estómago. En ambos casos, la fortaleza del personaje, que la línea 5230 había reducido en 10 puntos, se restablecerá a su nivel anterior.

En el tercer caso, la fortaleza del personaje se reducirá aún más (de ser necesario), a −1, y se establecerá la bandera de la muerte en el número del personaje. Si el personaje no está comiendo la empanada, se llevan a cabo comprobaciones para ver si se han satisfecho o no otras condiciones de la trama. Usted podrá ver exactamente qué sucede recorriendo los distintos caminos a través del árbol. Observe el uso de un nudo aleatorio que se bifurca en tres direcciones para dar una posibilidad entre tres de que se imprima un mensaje, y los dos nudos terminales de la esquina inferior izquierda del árbol, que vuelven a saltar dentro del árbol para permitir más comprobaciones.

Sugerencias

A estas alturas usted ya no tendrá dificultad alguna para "leer" los diagramas de estructuras arborescentes y ver los procesos implicados y las condiciones que se comprueban. Pruebe de seguir el recorrido de los otros árboles que se muestran y decida por sí mismo cómo se podrían entrar en nuestro listado. Quizá le interese considerar las implicaciones de recorrer árboles con diferente número de bifurcaciones desde distintos nudos.



Este árbol decide si contribuir o no a la atmósfera del juego imprimiendo un mensaje relacionado con el personaje que se está manipulando

Caroline Clayton

Perfectamente legal

Desarrollaremos un método sencillo para comprobar la legalidad y la sintaxis de una fórmula entrada en nuestra hoja electrónica

Anteriormente vimos un procedimiento para escribir fórmulas de hoja electrónica denominado *notación polaca inversa*. Por cuanto concierne a nuestro programa de hoja electrónica, la conversión de series de fórmulas escritas "normalmente" (infijos) a notación polaca inversa ofrece dos ventajas. Comprobar la legalidad de las expresiones en polaca inversa es más fácil que comprobar la de infijos y, una vez comprobada, también son más fáciles de evaluar. Veremos de forma detallada cómo se pue-

Aquí mostramos cómo se puede comprobar la legalidad de una serie en polaca inversa.

Al digitar programas, la mayoría de los programadores de BASIC se habrán encontrado con el mensaje de error SYNTAX ERROR en numerosas ocasiones. Las causas más comunes de los errores de sintaxis son los errores de digitación (pulsar una tecla equivocada) y saltarse información vital. Cuando un usuario de hoja electrónica entra una fórmula en una celda, se pueden plantear los mismos problemas. Por ejemplo, podría faltar un paréntesis o no haber suficientes operandos para las operaciones a realizar (pruebe de añadir un número en un espacio en blanco). Existe un método simple para comprobar una serie en notación polaca inversa y asegurar que estén presentes los componentes correctos y por un orden sensato, lo que se ha dado en llamar una serie polaca "bien formada".

La comprobación de que una serie polaca inversa está bien formada se presta idealmente para una aplicación de ordenador. A los elementos que componen una fórmula se les asignan valores así:

- Los operadores binarios (p. ej.,+,−) toman el valor −1
- Los operadores unarios (p. ej., &) toman el valor 0
- Los operandos (p. ej., A2,27) toman el valor 1

Luego se trabaja con la serie polaca inversa con un elemento por vez, de derecha a izquierda; se va llevando un total para los valores de tipos de elementos reseñados arriba. Si cada subtotal es menor o igual que cero y el total final es uno, entonces la expresión es legal. Si algún subtotal es mayor que cero o el total final no es uno, entonces la expresión es incorrecta. El ejemplo ilustra el procedimiento.

Una vez que la serie de infijos, I$, se ha convertido a su equivalente en polaca inversa, P$, la subrutina de la línea 4700 puede realizar una comprobación para asegurar que P$ esté bien formada. Esta rutina va trabajando a través de P$, desde la *izquierda* de a un elemento por vez, colocando los valores de tipos de elementos adecuados (−1, 0 o 1) en una pila, ST(). Habiendo terminado con P$, en la línea 4800 el programa pasa a procesar la pila. Se toma de la pila, ST(), cada tipo de elemento de a uno por vez, y se suma a un total. Si en algún momento durante este proceso (antes de que se saque de la pila el elemento final) al subtotal se hace superior a cero, la rutina se aborta con un mensaje ERROR ANTES DE TERMINAR. Tras procesar la pila por completo, el total final debe ser 1 si la expresión está bien formada. De no ser así, se genera un mensaje de error ERROR AL TERMINAR.

Dado que los operandos de la expresión pueden tomar la forma de nombres de celdas, como A2 o

Cómo se comprueba la legalidad

Para comprobar si el proceso de una serie polaca está bien formado, tomemos un ejemplo sencillo. Si partimos de una fórmula para una celda como ésta:

(A2+A3)*(B2−B3)

entonces la versión en polaca inversa es:

A2A3+B2B3−*

Esta lista refleja cómo se puede explorar empezando desde la derecha, llevando subtotales de los valores de tipos de elementos a medida que avanza:

Elemento	Valor tipo de elem.	Subtotal
*	−1	−1
−	−1	−2
B3	1	−1
B2	1	0
+	−1	−1
A3	1	0
A2	1	1

Al explorar una serie polaca inversa desde la derecha, siempre hay que encontrar un operador antes de los operandos con los que se relaciona. Por tanto, el subtotal debe ser negativo o, si se han encontrado ambos operandos, cero. La excepción la constituye el total final (que debe ser uno), porque en una serie siempre hay un operando más que operadores binarios.

Si por alguna razón el operador * final faltara en el ejemplo anterior, entonces el método detectaría inmediatamente el error:

Elemento	Valor tipo de elem.	Subtotal
−	−1	−1
B3	1	0
B2	1	1**ERROR**
+		
A3		
A2		



Probando las rutinas

En esta etapa se pueden probar las rutinas de conversión a notación polaca inversa y de comprobación de legalidad introduciendo en el programa una sencilla alteración. Digite el listado que ofrecemos aquí y luego cambie la línea 4010 de modo que rece:

```
4010 LET I$="tu serie de infijos"
```

Después entre las siguientes instrucciones en modalidad inmediata:

```
GOSUB 3000:GOSUB 4000:PRINT P$
```

Sinclair Spectrum:

```
4700>REM ******************************
4701 REM * COMPROBAR POLACA INVERSA      *
4702 REM * BIEN FORMADA                  *
4703 REM ******************************
4705 GO SUB 4900
4710 LET P=0: LET SP=1: LET L1=1: LET U$""
4720 LET P=P+1: IF P > LEN (P$) THEN GO TO
     4800
4730 LET T$=P$(P)
4740 IF T$="+" OR T$="-" OR T$="*" THEN
     LET S(SP)=-1: LET G$(SP)= T$: LET
     SP=SP+1: GO TO 4720
4745 IF T$="/" OR T$="^" THEN LET S(SP)=-1:
     LET G$(SP)=T$: LET SP= SP+1: GO TO 4720
4747 IF L1=LP THEN GO TO 4720
4750 LET U$=E$(LI)
4751 FOR Z=1 TO LEN(U$)
4752 IF U$(Z)=" " THEN GO TO 4754
4753 NEXT Z
4754 LET U$=U$(1 TO Z-1)
4760 LET U$<> P$(P TO P-1+LEN (U$)) THEN
     LET U$="": GO TO 4720
4770 LET S(SP)=1: LET G$(SP)=U$
4775 LET SP=SP+1
4780 IF L1<LP THEN LET L1=L1+1
4785 LET U$""
4790 GO TO 4720
4800 REM *** PROCESAR PILA *****
4810 LET P=S(SP)
4820 FOR C=SP-1 TO 1 STEP -1
4830 LET P=P+ S(C)
4840 IF P> 0 AND C<>1 THEN PRINT AT
     20,0;"ERROR ANTES DE TERMINAR":
     RETURN
4850 NEXT C
4860 IF P<> 1 THEN PRINT " ERROR AL
     TERMINAR ": RETURN
4870 LET CP=SP-1: RETURN
4900 REM *** HACER LISTA DE OPERANDOS *****
4905 LET P=0: LET LP=1: LET T$="": LET U$=""
4910 LET P=P+1: IF P>LEN (I$)THEN GO
     TO 4995
4920 LET T$=I$(P)
4930 IF T$="+" OR T$="-" OR T$="*" OR
     T$="&" THEN GO TO 4960
4940 IF T$="/" OR T$="^" OR T$="(" OR T$=")"
     THEN GO TO 4960
4950 LET U$=U$+T$: TO TO 4910
4960 IF U$="" THEN GO TO 4910
4970 LET E$(LP)=U$: LET LP=LP+1
4980 LET U$=""
4990 GO TO 4910
4995 IF U$="" THEN RETURN
4997 LET E$(LP)=U$: LET LP=LP+1
4998 RETURN
```

Complementos al BASIC

BBC Micro:
Introduzca las siguientes modificaciones en la versión para el Commodore 64:

```
4840 IF P>1 AND C<>1 THEN PRINT
     TAB(0,22);"ERROR ANTES DE
     TERMINAR ":RETURN
4860 IF P<>1 THEN PRINT TAB(0,22);"
     ERROR AL TERMINAR ":RETURN
```

Amstrad CPC 464/664:
Introduzca las siguientes modificaciones en la versión para el Commodore 64:

```
4840 IF P>1 AND C<>1 THEN LOCATE 1,22:
     PRINT " ERROR ANTES DE
     TERMINAR ":RETURN
4860 IF P<>1 THEN LOCATE 1,22:PRINT"
     ERROR AL TERMINAR ":RETURN
```

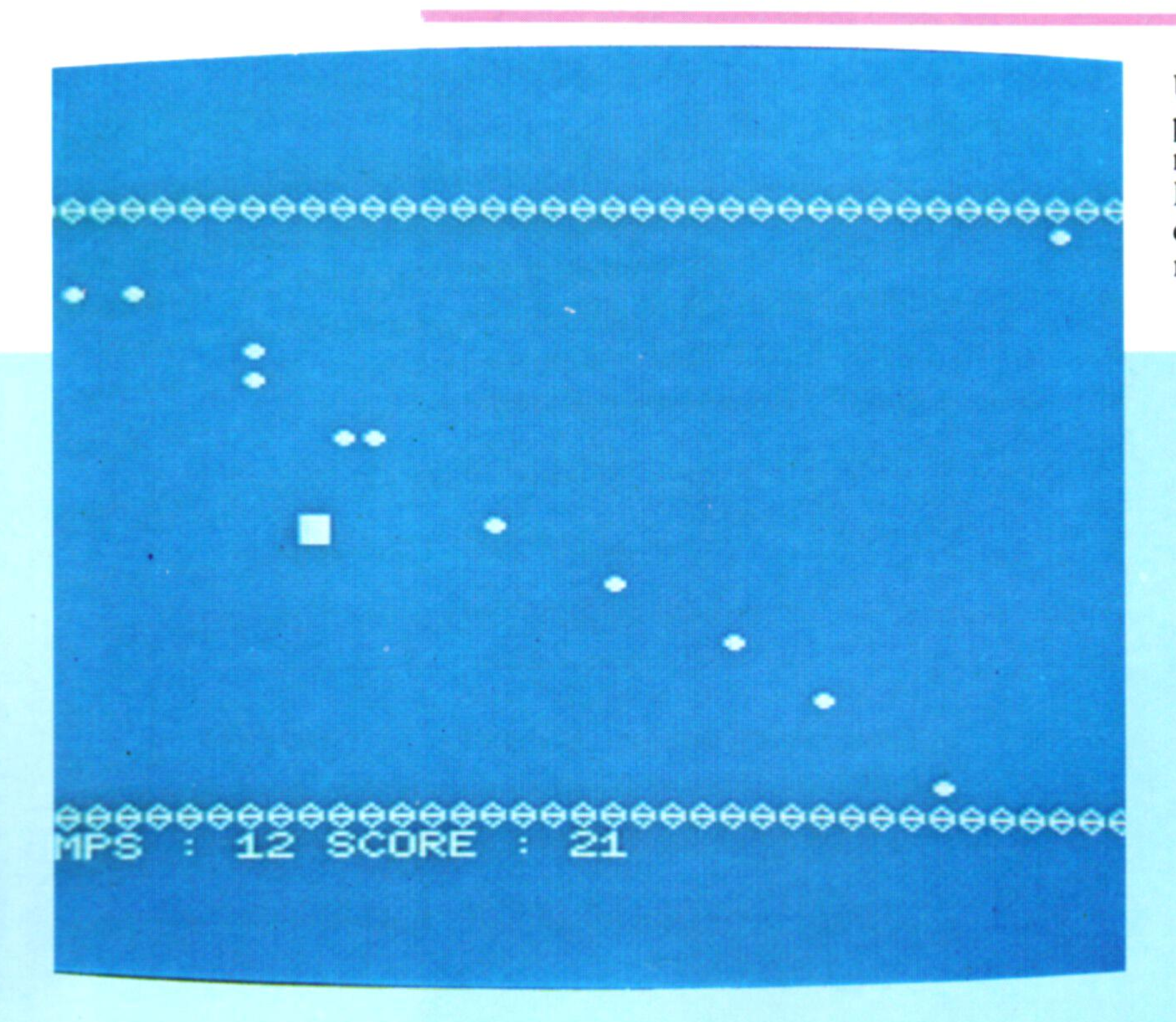
Commodore 64:

```
4700 REM*************************************
4701 REM * COMPROBAR SERIE POLACA            *
4702 REM * INVERSA BIEN FORMADA              *
4703 REM ************************************
4705 GOSUB 4900
4710 P=0: SP=1: L1=1: TE$=""
4720 P=P+: IF P> LEN(P$)THEN 4800
4730 T$=MID$(P$,P,1)
4740 IF T$="+" OR T$="-" OR T$="*" THEN
     ST(SP)=-1:G$(SP)=T$:SP=SP + 1: GOTO 4720
4745 IF T$="/" OR T$"^" THEN
     ST(SP)=-1:G$(SP)=T$:SP=SP+1:GOTO 4720
4746 IF T$="&" THEN ST(SP)=0:G$(SP)=T$:SP=SP+:GOTO
     4720
4747 IF L1=LP THEN 4720
4750 LET TE$=E$(L1)
4760 IF TE$<>MID$(P$,P,LEN(TE$)) THEN TE$="":GOTO
     4720
4770 LET ST(SP)=1:LET G$(SP)TE$
4775 LET SP=SP+1
4780 LET L1=L1-(L1<LP):TE$=""
4790 GOTO 4720
4800 REM *** PROCESAR PILA ***
4810 LET P=ST(SP)
4820 FOR C=SP-1 TO 1 STEP -1
4830 LET P=P+ST(C)
4840 IF P> 0 AND C <> 1 THEN GOSUB 1950:PRINT" ERROR
     ANTES DE TERMINAR ":RETURN
4850 NEXT C
4860 IF P<> 1 THE GOSUB 1950:PRINT " ERROR AL
     TERMINAR ": RETURN
4870 LET CP=SP-1:RETURN
4900 REM ** HACER LISTA DE OPERANDOS
     DE I$ **
4905 LET P=0:LET LP=1:LET T$="":LET TE$=""
4910 FOR K=1 T020:LET E$(K)="":NEXT K
4915 LET P=P+:IF P>LEN(I$) THEN 4995
4920 LET T$=MID$(I$,P,1)
4930 IF T$="+" OR T$="-" OR T$="*" OR T$="&" THEN
     4960
4940 IF T$="/" OR T$="^" OR T$="(" IR T$=")"
     THEN 4960
4950 LET TE$=TE$+T$:GOTO 4915
4960 IF TE$="" THEN 4915
4870 LET E$(LP)=TE$:LP=LP+1
4980 LET TE$=""
4990 GOTO 4915
4995 IF TE$="" THEN RETURN
4996 LET E$(LP)=TE$:LET LP=LP+1
4997 RETURN
```

L14, es importante aislarlos antes de comprobar la legalidad. Puesto que es más fácil identificarlos mientras la expresión está en su forma de infijos (porque todos los operandos están separados por operadores), la subrutina de la línea 4900 trabaja con la versión de infijos de la fórmula, I$, almacenando los nombres de operandos que vaya encontrando en E$( ).

Recogedor

Es posible que usted ya conozca este juego, que propone una curiosa forma de utilizar su ordenador. La versión que presentamos está destinada a los ordenadores MSX



Usted ha de intentar recoger lo más rápidamente posible las migajas que recubren el mantel. Dispone de 30 segundos para conseguir su limpieza total. Las migajas están representadas por puntos blancos. Las teclas de control del cursor, permitirán dirigir su recogedor.

```
10 REM ************************
20 REM *      RECOGEDOR       *
30 REM ************************
40 KEY OFF
50 GOSUB 260
60 LOCATE 0,23,0
70 PRINT "TIEMPO:";INT(Z);
   "PUNTUACION: ";S;
80 IF Z<1 THEN 580
90 K=STICK(0)
100 D1=(K=7) - (K=3)
110 D2=(K=1) - (K=5)
120 IF D1<>0 THEN DX=D1:DY=0
130 IF D2<>0 THEN DY=D2:DX=0
140 XP=PX+DX
150 YP=PY+DY
160 CR=VPEEK (XP+YP*40)
170 IF CR=J OR CR=B THEN XP=PX:YP=PY
180 IF CR=M THEN S=S+1:BEEP:X=X+1
190 VPOKE PX+PY*40,N
200 VPOKE XP+YP*40,J
210 PX=XP
220 PY=YP
230 Z=Z-.1
240 IF X=NM THEN 720
250 GOTO 60
260 SCREEN 0,0
270 COLOR 15,4,5
280 DEFINT A-Y
290 B=233
300 M=249
310 N=32
320 S=0
330 NM=10
340 X=0
350 FOR PX=0 TO 39
360 VPOKE PX+40,B
370 VPOKE 880+PX,B
380 NEXT PX
390 FOR PY=2 TO 21
400 VPOKE PY*40,B
410 VPOKE PY*40+39,B
420 NEXT PY
430 FOR I=1 TO NM
440 PX=INT (RND(-TIME)*37)+1
450 PY=INT (RND(-TIME)*21)+2
460 IF VPEEK(PX+PY*40)<>32 THEN 440
470 VPOKE PX+PY*40,M
480 NEXT I
490 PX=INT (RND(-TIME)*37)+1
500 PY=INT (RND(-TIME)*21)+2
510 IF VPEEK(PX+PY*40)<>32 THEN 490
520 VPOKE PX+PY*40,J
530 Z=30
540 DX=0
550 DY=0
560 J=219
570 RETURN
580 FOR I=1 TO 500
590 NEXT I
600 IF INKEY$<>""THEN 600
610 IF S>R THEN R=S
620 LOCATE 13,10
630 PRINT "RECORD:";R;
640 LOCATE 13,16
650 PRINT "OTRA?";
660 D$=INKEY$
670 IF D$="" THEN 660
680 IF D$<>"N" AND D$<>"n" THEN 50
690 CLS
700 LOCATE 0,0,1
710 END
720 NM=NM+1
730 VPOKE XP+YP*40,N
740 GOSUB 340
750 GOTO 60
```



Gestor eficiente

La firma Amstrad confía en que su flamante micro CPC 6128 se introduzca con buen pie en el importante mercado de gestión



Chris Stevens

Tras los pasos del éxito
El ordenador CPC 6128 de Amstrad sigue las huellas de las celebradas máquinas CPC 464 y 664, manteniendo la compatibilidad de software con sus predecesoras e incrementando al mismo tiempo su memoria a 128 Kbytes. La versión mejorada de CP/M que viene con el 6128 permite ejecutar paquetes de gestión CP/M clásicos

Un año después del lanzamiento del CPC 464, Amstrad ha captado el 25% del mercado de ordenadores personales de Gran Bretaña. El éxito de esta máquina basada en cassette se basó en el mismo concepto que los productos de *hi-fi* de Amstrad: empaquetar juntos todos los componentes del sistema, proporcionando las facilidades que desean la mayoría de los usuarios y vendiendo la unidad a un precio competitivo. El siguiente micro de Amstrad fue una versión mejorada del CPC 464, que incluía una unidad de disco integral y un BASIC ligeramente ampliado. El CPC 664 y la unidad de disco accesoria DDI-1 para el 464 configuraban CP/M y una versión del Dr LOGO de Digital Research. Pero apenas unos meses después Amstrad lanzaba el CPC 6128, una máquina similar en muchos sentidos al 664, pero con 128 K de RAM. El ordenador 6128 posee un aspecto más serio. Ha sido despojado de las teclas de control de color, que se han reemplazado por un elegante teclado gris uniforme. En los CPC 464/664 se suministraba un relleno numérico separado y un racimo de teclas para el cursor. En el 6128 estas teclas se han integrado en un único banco, haciendo que la anchura general de la nueva máquina sea unos 7 cm menor que la del 464. También se ha reducido la altura de la carcasa, de modo que ahora las teclas descansan a una altura que hace más cómoda la digitación. Además, las teclas poseen menor recorrido, proporcionando al digitar un tacto mucho más seguro.

Al igual que en el CPC 664, en la parte posterior de la máquina se proporcionan puertas para segunda unidad de disco, ampliación e impresora Centronics; pero las puertas para cassette, palanca de mando y audio se han desplazado hacia el lado izquierdo de la carcasa, mientras que los mandos de alimentación y de volumen se han trasladado del lado derecho a la parte posterior. La unidad de disco es la de 3 pulgadas de una sola cara utilizada en el CPC 664, pero el mecanismo se ha alojado en una carcasa mucho más fina y tapado con gráficos de referencia de números para las teclas y colores de pantalla.

Con el CPC 6128 se suministran dos discos. El primero de ellos incluye en una cara una versión mejorada de CP/M, denominada CP/M Plus. La misma está apoyada con varias utilidades de programación, incluyendo un ensamblador, desensamblador y utilidades de manejo de discos, además de las utilidades CP/M habituales, como PIP y SUBMIT. El segundo disco tiene una versión de 48 K del Dr LOGO, que representa una notable mejora con respecto a la versión empaquetada del CPC 664 y del DDI-1, con más de 50 instrucciones adicionales y primitivas.

En la cara superior del segundo disco también se incluyen el programa GSX de Digital Research y una facilidad HELP (ayuda). GSX fue el precursor del entorno GEM de DR, aunque no se hizo muy popular entre las empresas de software. La idea del GSX es que proporciona una "interface para gráficos de dispositivo transparente", lo que significa que las rutinas gráficas escritas utilizando el GSX en una máquina basada en Z80 o 8086 se ejecutará bajo GSX en cualquier otra máquina de base similar.

Si bien la capacidad de direccionamiento de un procesador de ocho bits es de 64 K, el 6128, basado en el Z80A de ocho bits, de algún modo se las arregla para doblar esta cantidad. Ello es posible en virtud del proceso que se conoce como conmutación de bancos, en el cual se pueden "conectar" y "desconectar" distintas áreas de ROM y RAM de los 64 K de espacio de direccionamiento del procesador.

La utilidad BANKMAN del disco de sistemas añade instrucciones adicionales al BASIC, permitiéndole manipular la memoria extra. Los 128 K de RAM están acomodados en dos secciones de 64 K, pero los programas en BASIC sólo pueden ocupar una de estas secciones, impidiendo que en el 6128 se escri-

CP/M Plus y mucho más

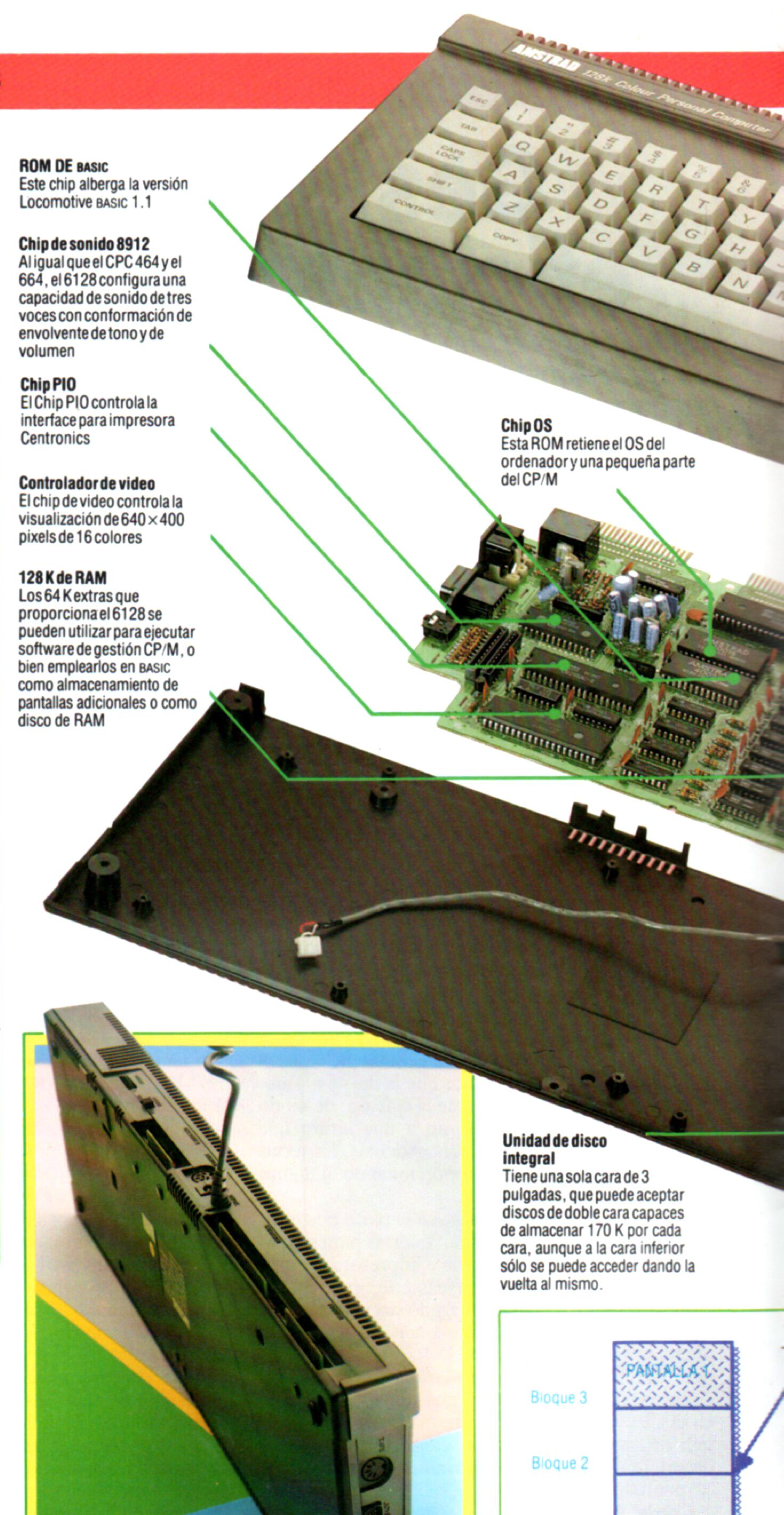
El CP/M Plus, la nueva versión de CP/M que se suministra con el CPC 6128, posee varias ventajas sobre el CP/M 2.2 utilizado en el CPC 664 y el 464 con unidad de disco DDI-1. El CP/M Plus emplea la memoria extra del 6128 para dejar libres 61,5 K de RAM para programas de aplicaciones una vez instalado. Esto es entre 3 y 4 K más de lo que se requiere para programas CP/M estándares y, por lo tanto, permite aprovechar toda una variedad de software de gestión CP/M disponible, incluyendo *SuperCalc*, *dBase II* y *WordStar*. Muchos paquetes CP/M están escritos para unidades de disco gemelas o de doble cara, e intentar acceder a una unidad que no esté presente da por resultado un error. El CP/M Plus se ha modificado para sortear este problema visualizando los mensajes de acción correctiva, de modo que el disco adecuado se pueda insertar de forma manual si así se requiriera. El CP/M Plus también incluye una facilidad para instalar juegos de caracteres de idiomas extranjeros y otra para actuar como terminal VT52 para un miniordenador u ordenador central

Espacio para trabajar

La versión mejorada del Dr LOGO suministrada con el CPC 6128 incluye instrucciones que estaban excluidas en la implementación original del CPC 664. Se le han incorporado instrucciones adicionales para procesamiento de listas, aritmética, gráficos, edición y disco, para hacer de ella una versión mucho más completa y útil. El tamaño global del espacio de trabajo ha pasado de los 2 105 nudos del CPC 664 a 3 753. Son de agradecer las características para acceder a una impresora y puertas E/S; a través de estas últimas es posible controlar tortugas móviles desde LOGO. Se incluyen instrucciones mejoradas de manipulación de disco que permiten cambiar el nombre de los archivos de un disco y seleccionar unidades alternativas desde el propio lenguaje. Instrucciones mejoradas de edición y depuración ofrecen la posibilidad de cargar archivos directamente desde disco en el editor de pantalla del LOGO, y listar en el espacio de trabajo variables globales y procedimientos

ban programas más amplios de los que podrían escribirse en el CPC 464 o 664. No obstante, la sección de 64 K que no se utiliza para programas se puede emplear como zona de almacenamiento a la que se puede acceder desde BASIC. Las nuevas instrucciones permiten utilizar esta memoria extra ya sea para almacenar visualizaciones en pantalla adicionales, que se pueden pasar a la zona de pantalla normal, o bien actuar como un sistema de archivo de registros.

Debido a que la visualización en pantalla exige 16 K, los 64 K de la sección extra pueden retener cuatro pantallas adicionales, numeradas del 2 al 5. La pantalla 1, la estándar, se retiene en la otra sección de 64 K utilizada por el BASIC. La instrucción SCREENCOPY,A,B copia la pantalla B en la zona de 16 K de la pantalla A, escribiendo sobre el contenido original, mientras que SCREENSWAP,A,B intercambia los contenidos de las dos zonas de pantalla.



ROM DE BASIC
Este chip alberga la versión Locomotive BASIC 1.1

Chip de sonido 8912
Al igual que el CPC 464 y el 664, el 6128 configura una capacidad de sonido de tres voces con conformación de envolvente de tono y de volumen

Chip PIO
El Chip PIO controla la interface para impresora Centronics

Controlador de video
El chip de video controla la visualización de 640 × 400 pixels de 16 colores

128 K de RAM
Los 64 K extras que proporciona el 6128 se pueden utilizar para ejecutar software de gestión CP/M, o bien emplearlos en BASIC como almacenamiento de pantallas adicionales o como disco de RAM

Chip OS
Esta ROM retiene el OS del ordenador y una pequeña parte del CP/M

Unidad de disco integral
Tiene una sola cara de 3 pulgadas, que puede aceptar discos de doble cara capaces de almacenar 170 K por cada cara, aunque a la cara inferior sólo se puede acceder dando la vuelta al mismo.

Accesos para puertas
El 6128 configura puertas para impresora Centronics, ampliación, segunda unidad de disco, palanca de mando, sonido estéreo y cassette. También hay presentes conexiones para 5 V, 12 V y pantalla, y se acoplan a la pantalla proporcionada



ULA
Este chip construido a medida sincroniza las operaciones internas, incluyendo la conmutación de bancos de memoria y la temporización de pantalla

Teclas de función y del cursor
Para reducir las dimensiones globales de la unidad, Amstrad ha trasladado las teclas de función y del cursor más cerca del teclado principal. Ahora la tecla Return está rodeada por otras, con lo que aumentan las posibilidades de cometer errores al digitar

Controlador de disco flexible
Este chip maneja la unidad de disco integrada y la segunda unidad opcional

Chris Stevens

Un intercambio de pantalla puede emplear medio segundo, pero es posible intercambiar o copiar una 64ava parte de la pantalla por vez. Es probable que los escritores de juegos aprovechen cabalmente estas facilidades para cambios rápidos en pequeñas porciones de la pantalla.

La sección de 64 K extra también se puede utilizar como un archivo, en cuyo caso la memoria extra se dispone en varios registros donde se pueden almacenar series en BASIC: el disco de RAM, así llamado en razón de su similitud con la estructura de un archivo en disco. Todos los registros del archivo deben tener la misma longitud, que se establece mediante la sentencia IBANKOPEN,n, donde n es la longitud de cada registro, de entre 2 y 255 caracteres. IBANKREAD e IBANKWRITE permiten pasar los datos en serie desde o hacia el disco de RAM, y con estas instrucciones se puede pasar un número de registro opcional que especifica el registro a utilizar. Si no se especifica ningún número de registro, el BASIC comenzará a escribir o a leer empleando el último registro especificado (o el primer registro) e incrementará automáticamente un señalador listo para ocuparse del siguiente registro. En otras palabras, se puede crear y utilizar tanto un archivo de acceso aleatorio como uno secuencial.

El CPC 6128 es una máquina compacta y estilizada que ofrece una buena relación precio-prestaciones. La principal crítica que se le hizo al CPC 664 era que, si bien se suministraba con CP/M, no tenía memoria suficiente como para considerarlo un ordenador serio para uso en pequeña gestión. La ampliación de la RAM a 128 K, junto con una versión de CP/M mejorada capaz de utilizar la memoria extra, le permiten al 6128 ejecutar muchos paquetes CP/M estándares. Asimismo, la nueva máquina también es compatible, desde el punto de vista del software, con el CPC 464 y el 664 y, en consecuencia, dispone de numerosos paquetes elaborados a su medida. Con estos atributos, el CPC 6128 representa una propuesta muy atractiva para el usuario que desee una máquina de juegos y una máquina de gestión, todo en una.

Cambio de dirección

PANTALLA 5 — Bloque 7
PANTALLA 4 — Bloque 6
PANTALLA 3 — Bloque 5
PANTALLA 2 — Bloque 4
Segunda sección de 64 K

El BASIC suministrado con el 6128 no puede acceder a la totalidad de los 128 K de RAM, y sólo puede trabajar con programas comprendidos en la primera sección de 64 K. Los otros 64 K se pueden utilizar como un disco de RAM o para almacenar cuatro visualizaciones alternativas. Ya que el procesador Z80A sólo puede direccionar 64 K por vez, la memoria extra se divide en bloques de 16 K, de modo que cualquier bloque se puede conmutar temporalmente entre las direcciones &4000 y &7FFF

AMSTRAD CPC 6128

MEMORIA

128 K de RAM, 48 K de ROM. Sólo hay 64 K disponibles para programas en BASIC, pero los otros 64 K se pueden utilizar como disco de RAM o para almacenar hasta cuatro visualizaciones de pantalla alternativas

CPU

Z80A trabajando a 4MHz

DISCO

Una unidad de tres pulgadas de una sola cara, con una puerta para una segunda unidad

INTERFACES

Puertas para bus de ampliación, interface para segunda unidad de disco flexible, cassette, estéreo, palanca de mando e impresora Centronics, entradas de 12 V y 5 V, conector para pantalla

SOFTWARE SUMINISTRADO

Dos discos que contienen CP/M Plus, utilidades de programación, Dr LOGO mejorado y GSX. Asimismo, se proporcionan CP/M 2.2 y Dr LOGO reducido por razones de compatibilidad con el 664 y el 464

DOCUMENTACION

El manual de instrucciones para el usuario cubre la programación en BASIC y las ampliaciones de gestión de memoria al sistema. Las secciones dedicadas al Dr LOGO y al CP/M cubren los aspectos básicos de operación, pero Amstrad ofrece manuales más detallados sobre estos temas

VENTAJAS

El CPC 6128 representa el límite de la tecnología del ordenador de 8 bits y su relación precio-prestaciones es excelente. Su capacidad para ejecutar paquetes CP/M estándares, junto con su excelente BASIC y gráficos, lo hacen igualmente útil para aplicaciones de gestión y de juegos

DESVENTAJAS

Amstrad ha vuelto a utilizar unidades de 3 pulgadas en vez de las unidades de 3½ pulgadas que emplean muchas máquinas de pequeña gestión. Debido a que las unidades son de una sola cara y que algunos programas utilizan más de una cara de un disco, es probable que los usuarios hayan de ocupar cierto margen de tiempo en manipular discos

Adaptación musical

Adaptaremos la interface para que pueda ser utilizada con la gama Amstrad CPC

Lista de componentes

N.°	Artículo
1	condensador policarbonato 1 nF
2	condensador policarbonato 100 nF
1	resistencia 270 ohmios
3	resistencia 220 ohmios
2	resistencia 680 ohmios
2	conectores para montar en la placa, DIN de 5 patillas y 180°
1	chip ACIA MC68B50
1	chip aislador óptico 6N139
1	inversor héxuple TTL 74LS04
1	puerta NAND triple de tres entradas 74LS10N
1	zócalo DIL 24 patillas
2	zócalo DIL 14 patillas
1	zócalo DIL 8 patillas
1	cristal 2.0 MHz
1	diodo 1N914
2	conectores de borde de placa de 50 vías
30 cm	cable plano de 50 vías

La placa DIP que se utiliza para montar los componentes se puede conseguir en casi todas las sucursales Tandy, bajo el número de componente 276-164.

El diseño de hardware de la interface para el Amstrad es básicamente el mismo que el de la versión para el BBC Micro y el Commodore 64. Las principales diferencias provienen del hecho de que la máquina Amstrad se basa en el procesador Z80 en vez del 65XX que utilizan las máquinas del diseño original. En consecuencia, el bus del Z80 no es compatible con el dispositivo ACIA MC6850 empleado en la interface.

El circuito del bus de la interface requiere un componente adicional: un IC que contenga tres puertas NAND de tres entradas. También se requieren las dos líneas *select* del chip ACIA, CS0 y CS1 (en el circuito original estaban conectadas directamente a +5 V). La línea *enable* del chip ACIA es básicamente lo inverso de la línea $\overline{\text{IORQ}}$ (*I/O request*: solicitud de E/S) del Z80, pero está sincronizado por M1 para impedir el acceso E/S durante la secuencia de conocimiento de interrupción del Z80, en el curso de la cual $\overline{\text{IORQ}}$ y $\overline{\text{M1}}$ se ponen bajos simultáneamente. También está sincronizado por la dirección A7.

La puerta para ampliación del Amstrad no proporciona una línea de decodificación idónea para la conexión directa a dispositivos E/S externos y, por tanto, la decodificación se ha de hacer externamente. El direccionamiento E/S en la gama CPC utiliza la totalidad de los 16 bits de direcciones (durante la mayoría de las instrucciones E/S del Z80, los contenidos del registro B salen en los ocho bits de direcciones superiores). El bus de direcciones dispone para su uso las direcciones desde &F800 a &FBFF, mientras que los ocho bits inferiores han de estar entre &E0 y &FE para los periféricos del usuario.

Esto no es tan complicado como parece, porque todas las direcciones E/S internas tienen el bit de direcciones A10 en +5 V. Por consiguiente, para decodificar nuestra gama de direcciones necesitamos detectar un nivel bajo en A10, y niveles altos en A5-A7. Esto se efectúa conectando A10 a CS2, y A5 y A6 a CS0 y CS1 respectivamente, mientras que la dirección A7 se utiliza para sincronizar la señal *enable* (E), impidiéndole ponerse activa a menos que A7 se ponga alto.

Lectura y escritura

A diferencia del procesador 6502, el Z80 no proporciona una única señal de lectura/escritura, de modo que nuestro diseño trata a los registros de lectura y escritura del 6850 como direcciones diferentes, uniendo la línea R/W a la línea de direcciones A9. La línea *select* del registro interno se conecta a A8, lo que da por resultado que a los registros ACIA se les asignen las direcciones E/S, como refleja la tabla de ubicación de registros ACIA (ver p. 2114).

Es importante observar que, debido a que la acción de leer o escribir en el ACIA está controlada por una línea de direcciones, no es posible escribir en las direcciones de registros de lectura solamente (&FAE0, &FBE0). Esto dará por resultado que tanto el Z80 como el ACIA intenten simultáneamente colocar un byte de datos en el bus del sistema. Es todavía más peligroso intentar leer las direcciones de sólo escritura (&F8E0, &F9E0). Ello daría por resultado que se efectuara una escritura en el ACIA con el bus de datos en un estado flotante impredecible.

La placa se debe comprobar del mismo modo que se comprobó la versión anterior. Aquí presentamos nuevamente el procedimiento, con las regulaciones adecuadas para el Amstrad.

1. Conectar un cable DIN de 5 patillas estándar entre los conectores IN y OUT de la placa.
2. Inicializar el ACIA con la siguiente instrucción:

```
OUT &F8E0,3
```



Disposición de los componentes

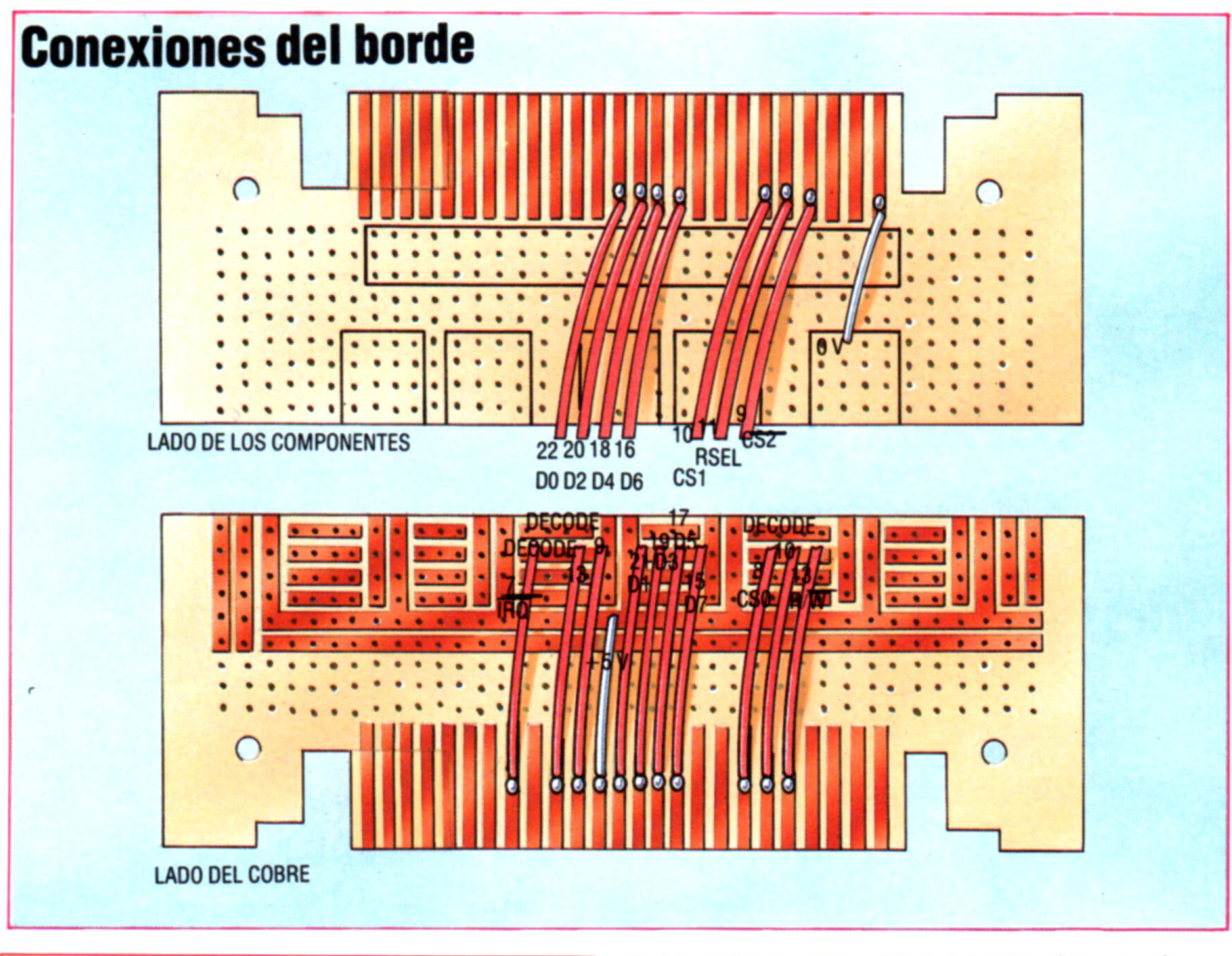
ENTRADA MIDI
SALIDA MIDI
CONECTORES 180° 5 PATILLAS
RESISTENCIAS 220 OHMIOS
DIODO 1N914
CHIP AISLADOR ÓPTICO 6N139
RESISTENCIA 270 OHMIOS
CHIP PUERTAS NAND
CONDENSADOR POLICARBONATO 1 nF
RESISTENCIAS 680 OHMIOS
CHIP ACIA MC68B50
CHIP INVERSOR HÉXUPLE 74LS04
CONDENSADOR POLICARBONATO 100 nF
LADO DE LOS COMPONENTES

ESTOS GRUPOS DE PATILLAS ADYACENTES SE DEBEN CONECTAR EN LA CARA INFERIOR DE LA PLACA
CHIP TRIPLE PUERTAS NAND
CHIP AISLADOR ÓPTICO
CHIP INVERSOR HÉXUPLE
CHIP ACIA
LADO DEL COBRE

Kevin Jones

Detalles de montaje
Los componentes principales de la interface MIDI deben montarse en la placa especial. Comience por montar los componentes pasivos: las resistencias, los condensadores, los conectores DIL y DIN. Realice las conexiones necesarias en la placa utilizando cable de arrollar aislado y monte el cristal. Asegúrese de que el diodo quede montado de modo que el extremo marcado con una franja de color quede arriba, y no olvide la pequeña unión debajo del condensador situado más cerca del chip inversor héxuple

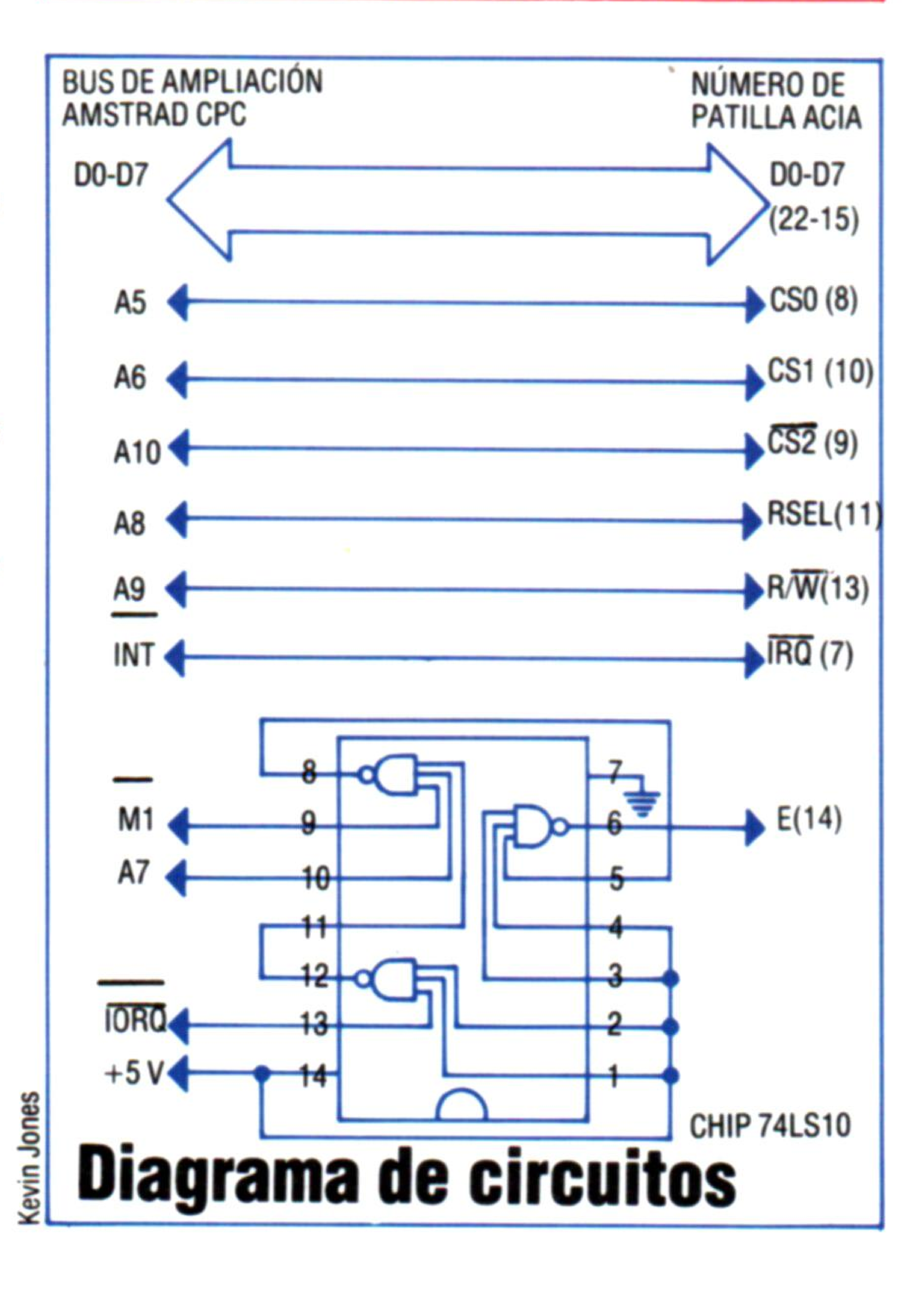
Conexiones del borde
El extremo izquierdo de la placa se utiliza para aceptar señales desde el bus de ampliación Amstrad. Estos diagramas muestran las conexiones de bus adecuadas con el chip ACIA, el chip decodificador 74LS10 y las alimentaciones. Cuando se hayan realizado todas estas conexiones con cable de arrollar aislado se pueden empujar suavemente los cuatro ICs en sus zócalos, teniendo cuidado de respetar la orientación correcta de las muescas. Por último, construya un cable de 50 vías utilizando dos conectores de borde de 50 vías y un trozo de 30 cm de cable plano. Presione un conector en la placa y el otro en la puerta para ampliación de su Amstrad, asegurándose de que el ordenador esté apagado. Ahora la placa está terminada y lista para probarla



Ubicación de registros ACIA

Dirección	Registro	
&F8E0	Control	(SÓLO ESCRITURA)
&F9E0	Transmisión datos	(SÓLO ESCRITURA)
&FAE0	Estado	(SÓLO LECTURA)

Decodificación Amstrad
Nuestro diseño de circuito original de la interface MIDI estaba pensado para máquinas basadas en el 65XX. Para usar la interface con una máquina basada en el Z80, como cualquiera de la gama Amstrad CPC, se han de decodificar las señales del bus del ordenador antes de la conexión con el chip ACIA. Este diagrama de circuitos muestra la lógica de decodificación necesaria que se obtiene en nuestro diseño corregido utilizando un solo chip triple de puertas NAND de tres entradas



3. Establecer la velocidad del reloj, formato de palabras en serie e inhabilitar interrupciones recibir/transmitir digitando:

```
OUT &F8E0,&16
```

4. Leer el registro de estado del ACIA mediante:

```
PRINT INP(&FAE0)
```

El valor correcto debe ser dos, que indica que la conexión del bus está funcionando correctamente.
5. Enviar un byte en serie desde la salida a la entrada digitando:

```
OUT &F9E0,x
```

donde x puede ser cualquier entero entre cero y 255.
6. Verificar la recepción correcta del byte volviendo a leer el registro de estado:

```
PRINT INP(&FAE0)
```

Esto habrá de devolver el valor tres.
7. Verificar, asimismo, que el byte sea el valor esperado, leyendo el registro de datos de recepción:

```
PRINT INP(&FBE0)
```

Esto devolverá el mismo valor x que se utilizó en el paso 5. Los pasos del 4 al 7 se pueden repetir tantas veces como sea necesario utilizando distintos valores de x.

Si alguna de estas pruebas falla, o si el ordenador se queda colgado con la placa conectada, compruebe el cableado con un tester, en caso de que disponga de uno. Si aun así los problemas persisten, remítase a la p. 1848, donde analizamos algunos de los problemas más probables.

En el próximo y último capítulo de este proyecto crearemos software para nuestra interface.



La dimensión FORTRAN

Veamos de qué manera el FORTRAN emplea las estructuras de matriz

La única estructura de datos de que dispone el programador de FORTRAN es la matriz y, al igual que en el BASIC, debe declararse antes de utilizarla; dado que el FORTRAN es un lenguaje compilado, esto ha de hacerse justo al comienzo del programa. Esto se realiza por medio de una sentencia DIMENSION, que funciona de la misma manera que una sentencia DIM del BASIC.

Utilizando los tipos de datos por defecto, en los cuales toda variable cuyo nombre empiece con I, J, K, L, M o N es de enteros y todas las demás son reales, una sentencia como:

```
DIMENSION A(20),I(50)
```

reservará espacio de memoria para una matriz de hasta 20 números reales y otra matriz de hasta 50 números enteros. Si se quiere pasar por alto las tipologías por defecto con sentencias INTEGER, REAL, DOUBLE PRECISION, COMPLEX o LOGICAL, entonces se debe dar uno de éstos además de la sentencia DIMENSION. Sin embargo, se puede hacer la declaración de la matriz mientras se efectúa la declaración del tipo:

```
INTEGER ARR1(20)
REAL NUMS(30)
```

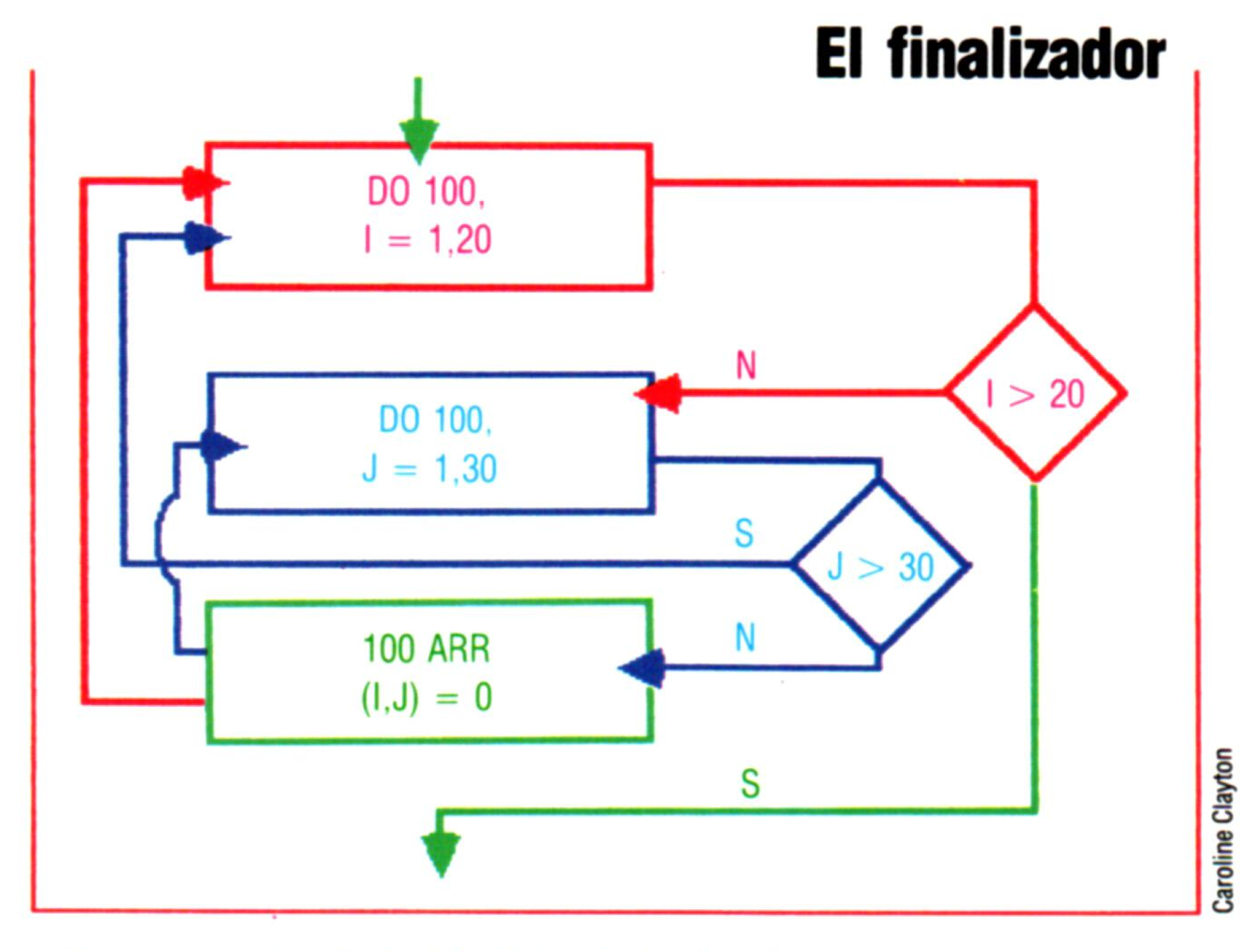
Esto reserva espacio para una matriz de 20 enteros y una para 30 números reales.

Son factibles matrices bidimensionales y algunas versiones permiten tres o más dimensiones. Las matrices bidimensionales se declaran de la forma normal. He aquí dos ejemplos:

```
DIMENSION ARR(20,30)
INTEGER ARR(20,30)
```

Para indexar la matriz se puede utilizar cualquier constante o variable de enteros, como ARR(3,4) o INTARR(I). Aunque en FORTRAN IV no existe nada que se parezca a una serie de caracteres, esta deficiencia, junto con otras cosas, se palió en el FORTRAN 77, que veremos con mayor profundidad en el próximo capítulo. Los caracteres se almacenan como enteros en variables de enteros comunes; en el último capítulo vimos cómo se podían almacenar uno o más caracteres en cualquier variable de enteros. En FORTRAN, lo más cercano a una serie de caracteres es una matriz de enteros, pero esto no es realmente satisfactorio para quien esté acostumbrado a las amplias facilidades de manipulación de series que ofrecen casi todos los BASIC.

El finalizador



Código compacto
El FORTRAN carece de alguna sentencia clara de fin del bucle, y más de un bucle puede compartir la misma sentencia de terminación. Esto puede dar origen a una codificación muy compacta, como vemos en este ejemplo de flujo de programa en el que dos bucles comparten el mismo finalizador. El equivalente de BASIC, aunque menos compacto, es mucho más fácil de leer y, por tanto, de depurar:

```
100 FOR I=1 TO 20
110 FOR J=1 TO 30
120 LET A (I,J)=0
130 NEXT J
140 NEXT I
```

En FORTRAN, la principal facilidad de bucle y la única estructura de control (aparte de las sentencias IF y GOTO) es el bucle DO, que efectivamente opera en el mismo estilo que un bucle FOR...NEXT de BASIC. El bucle DO asume la forma:

```
   DO Sn(var. entera)= val.inicial,
      val.final, val.paso
   .....
Sn (última sentencia del bloque a repetir)
```

Sn es el número de la última sentencia del bloque que se ha de repetir. Como contador del bucle (var. entera), se puede utilizar cualquier variable de enteros, y se especifican los valores de comienzo, final y paso. En BASIC, el bucle DO sería así:

```
FOR var. entera=val.inicial TO val.final STEP
    val.paso
.....
NEXT var. entera
```

Para hacer que el valor de una matriz sea cero, por ejemplo, entraríamos:

```
    DO 100 I = 1,20,1
100 ARR(I)=0
```

cuando el valor del paso de una matriz es 0, entonces se puede omitir, como en:

```
    DO 100 I = 1,20
100 ARR(I)=0
```

La sentencia que se numera para marcar el final del bucle puede ser cualquier sentencia ejecutable, con la excepción de una IF, una GOTO u otra DO.

El hecho de que no haya ninguna sentencia clara de final del bucle a veces resulta útil si se quiere producir un código más compacto, pero también puede conducir a algunos errores garrafales, así como a un código que resulte absolutamente incomprensible incluso al propio autor, en especial cuando hay dos o más bucles anidados uno dentro de otro. Por ejemplo, esta inicialización de una matriz bidimensional es perfectamente legal:

```
      DO 100 I=1,20
      DO 100 J=1,30
100   ARR (I,J)=0
```

(Observe que en ambos bucles se ha utilizado como finalizador la misma sentencia.)

Para superar las restricciones sobre la sentencia final de un bucle DO y obtener un código más claro, con frecuencia se emplea la sentencia "ficticia" CONTINUE a modo de finalizador. Se trata de una sentencia de FORTRAN ejecutable y legal que no hace absolutamente nada y se puede utilizar en cualquier lugar de un programa donde se necesite. La convención normal exige terminar los bucles DO con su propio y exclusivo CONTINUE, de modo que el ejemplo anterior se convertiría en:

```
      DO101 I=1,20
        DO100 J=1,30
          ARR(I,J)=0
100     CONTINUE
101   CONTINUE
```

Algunos programadores llegan aún más lejos e insisten en que todas las transferencias de control, como los DO, IF y GOTO, han de terminar con un CONTINUE, lo que ciertamente ayuda a hacer que los programas en FORTRAN resulten más comprensibles.

Como lenguaje matemático, cabe esperar que el FORTRAN proporcione algunas facilidades adicionales para la manipulación de matrices de números, particularmente en entrada/salida. Toda matriz uni o bidimensional se puede leer o escribir en una única sentencia READ o WRITE, donde el FORMAT correspondiente da la posición de cada línea o registro. Por ejemplo:

```
      DIMENSION IARR1(20)
      .....
      WRITE(1,10)IARR1
```

produciría la salida de todo el contenido de IARR1 (20 valores de entero). Además:

```
      10 FORMAT (20I4)
```

produciría 20 enteros de cuatro dígitos a lo largo de una línea;

```
      10 FORMAT (I4)
```

daría uno por línea;

```
      10 FORMAT (5I4)
```

daría cuatro líneas de cinco números, y:

```
      10 FORMAT (20A1)
```

daría una serie de 20 caracteres en una línea.

Existe un problema cuando se entran o se sacan matrices bidimensionales de este modo: el FORTRAN, a diferencia de casi todos los otros lenguajes, almacena las matrices bidimensionales por orden de columna en lugar de por orden de fila. Si usted no tiene cuidado, se encontrará trabajando con una versión transpuesta de lo que realmente quería. Para sortear este problema, y para situaciones en las que no está manejando la totalidad de una matriz, se proporciona una forma especial de DO denominada *bucle* DO *implícito*, que se escribe con el nombre de la variable en la sentencia READ o WRITE. Por ejemplo, para rellenar IARR2(10,20) con 10 series de 20 caracteres, se podría usar:

```
      DO 100 I=1,10
      READ (1,10)(IARR2 (I,J),J=1,20)
10    FORMAT (20A1)
100   CONTINUE
```

El FORTRAN posee una auténtica facilidad, aunque algo limitada, de subrutinas. Éstas son completamente independientes y suelen emplear sólo variables locales, y por lo general pueden añadirse al final de un programa principal o escribirlas y compilarlas por separado. Es corriente utilizar los mismos números de línea nuevamente en una subrutina, y en la mayoría de los casos los mismos nombres de variables, sin riesgo de confusión. El paso de parámetros es sólo por referencia; en otras palabras, la dirección del parámetro se pasa a la subrutina cuando se llama, de modo que cualquier cambio de valor provocado por la subrutina afectará a la variable correspondiente en el programa de llamada. Las variables globales, a las que se puede hacer referencia en la subrutina y en el programa principal, se pueden declarar, si así se requiriera, mediante una sentencia COMMON en ambos módulos del programa. Se pueden establecer varios bloques COMMON y asignarles un nombre, pero por lo general basta un solo bloque sin nombre.

Las variables designadas en las sentencias COMMON de dos módulos deben coincidir en el uso total de memoria, pero no tienen que coincidir en tipo, si bien no se recomienda esto como mecanismo para cambiar el tipo. Por ejemplo, un programa principal podría contener:

```
      COMMON I1,I2,I3
```

donde la subrutina podría contener:

```
      COMMON I(3)
```

El uso de memoria es el mismo en ambos casos; pero en el primero, las tres variables de enteros se mantienen distintas, mientras que en el segundo se alude a ellas como elementos de una matriz.

Un subrutina típica podría tener la forma:

```
      CALL MYSUB (X,Y,I,J)
```

donde la subrutina comenzaría con una sentencia:

```
      SUBROUTINE MYSUB (A,B,L,M)
```

(Observe que los parámetros dentro de los paréntesis deben coincidir en número y tipo.)



El control retorna desde la subrutina al programa de llamada mediante una sentencia RETURN (de hecho, en una subrutina puede haber más de un RETURN). Uno de éstos se debe ejecutar independientemente del camino de control que se siga a través de la subrutina. La sentencia final de un subprograma de subrutinas debe ser un END, aunque ésta es más una directriz para el compilador que una sentencia ejecutable.

Una facilidad interesante del paso de parámetros es que, en cierto sentido, es posible cambiar el tamaño de una matriz dinámicamente en tiempo de ejecución. En un lenguaje interpretado como el BASIC esto se hace fácilmente, pero es muy raro en un lenguaje compilado, en el cual el espacio de almacenamiento se debe asignar en tiempo de compilación en vez de en tiempo de ejecución.

Además de las subrutinas, el FORTRAN proporciona una facilidad FUNCTION para los subprogramas, que devuelve un único valor y que en muchos sentidos es similar a la del PASCAL. Un subprograma de función se escribe por separado, de modo similar a una subrutina, pero empezando con una sentencia FUNCTION. Antes de que se ejecute un RETURN se debe efectuar una asignación al nombre de función. El tipo de valor por defecto que devuelve la función está determinado por el nombre de función utilizando la convención habitual. De ser necesario se podría obviar este requisito, aunque ello no es recomendable a menos que lo requiera una función empleada por varios programas diferentes. Las funciones se pueden utilizar en el programa principal del mismo modo que las funciones de biblioteca (simplemente usando su nombre, con una lista de parámetros) en cualquier expresión en la que sea válido usar una variable de ese tipo.

Hallando las raíces

```
C    UN PROGRAMA EN FORTRAN PARA LEER
C    CONJUNTOS DE 3 VALORES A,B,C QUE REPR.
C    LOS COEFICIENTES DE UNA ECUACION DE 2°
C    GRADO A*X**2+B*X+ C=0 USANDO LA
C    FORMULA USUAL
     FUNCTION DISCR(A,B,C)
     DISC=B*B-4.0*A*C
     RETURN
     END
     SUBRUT.RESOLVER(A,B,C,X1,X2,RAIZN)
     D=DISCR(A,B,C)
C    OBSERVE QUE AQUI SE PODRIA UTILIZAR
C    UN IF ARITMETICO PERO NO ES ACONSEJABLE
     IF(D.GE.0.0)GOTO 100
C    D ES MENOR QUE 0 POR TANTO R. COMPL.
     RAIZN=0
     X1=0.0-B/(2.0*A)
     X2=SQR(ABS(D))
     GOTO 300
100  IF(D.GT.0.0)GOTO 200
C    D ES CERO DE MODO QUE RAICES IGUALES
     RAIZN=1
     X1=0.0-B/(2.0*A)
     X2=X1
     GOTO 300
C    D ES POSITIVO POR TANTO DOS RAICES
     RAIZN=2
200  X1=(0.0-B+ SQRT(D))/2.0*A
     X2=(0.0-B-SQURT(D))/2.0*A
300  RETURN
     END
C    PRIMERO LEER NUM. CONJUNTOS VALORES
     READ(2,10)VALN
C    IMPRIMIR TITULOS
     WRITE(3,11)
     DO 500 I=1, VALN
C    LEER TRES VALORES Y COMPROBAR
C    SI ACEPTABLES
100     READ(1,12)A,B,C
        IF (A.EQ.0.0)GOTO 500
C    LOS VALORES SON ADECUADOS DE MODO QUE
     LLAMAR SUBRUT.
        CALL SOLVE(A,B,C,X1,X2,RAIZN) IF(RAIZN.GT.0)GOTO 200
C    RAICES COMPLEJAS
        WRITE(1,13)A,B,C,X1,X2
        GOTO 500
200  IF(RAIZN.GT.1)GOTO 300
C    RAICES IGUALES
        WRITE(1,14)A,B,C,X1,X2
        GOTO 500
C    DOS RAICES
300  WRITE(1,15)A,B,C,X1,X2
        GOTO 500
C    VALORES NO VALIDOS PARA A,B Y C
400     WRITE (1,16)A,B,C
500  CONTINUE
     STOP
C    SENTENCIAS FORMAT
10   FORMAT(I4)
11   FORMAT(1H,8X,1HA,8X,1HB, 8X,
     1HC,8X,4NTIPO,8X,2HX1,8X,2HX2)
12   FORMAT(1H,3F7.2)
13   FORMAT(1H,4X,3(F7.2,2X),4X,7H
     COMPLEJA,X,F7.2,3X,F7.2)
14   FORMAT(1H,4X,3,(F7.2,2X),4X,5
     HIGUAL3X,F7.2,3X,F7.2)
15   FORMAT(1H,4X,3,(F7.2,2X),4X,7H
     DESIGUAL,X,F7.2,3X,F7.2)
16   FORMAT(1H,4X,3,(F7.2,2X),4X,7HNO
     VALIDA)
     END
```

Cálculo de media

```
C UNA FUNCION EN FORTRAN PARA
C CALCULAR LA MEDIA (REAL) DE UN
C CONJUNTO DE N NUMEROS ENTEROS
  EN UNA MATRIZ DE ENTEROS IARR
  FUNCION PROM(IARR,N)
  DIMENSION IARR(N)
  SUM=0
  DO 100 I=1,N
     SUM=SUM+FLOAT(IARR(I))
100 CONTINUE
  PROM=SUM/FLOAT(N)
  RETURN
  END
```

Perfecto pinchadiscos

El OS de los ordenadores Amstrad presenta facilidades sonoras que equivalen a las instrucciones en BASIC

El diseño hardware de los Amstrad emplea el conocido generador de sonidos programable AY-3-8912 de General Instruments (llamado brevemente *8912*), que no sólo sirve para generar sonidos sino que también puede rastrear el teclado mediante la puerta de siete bits del chip.

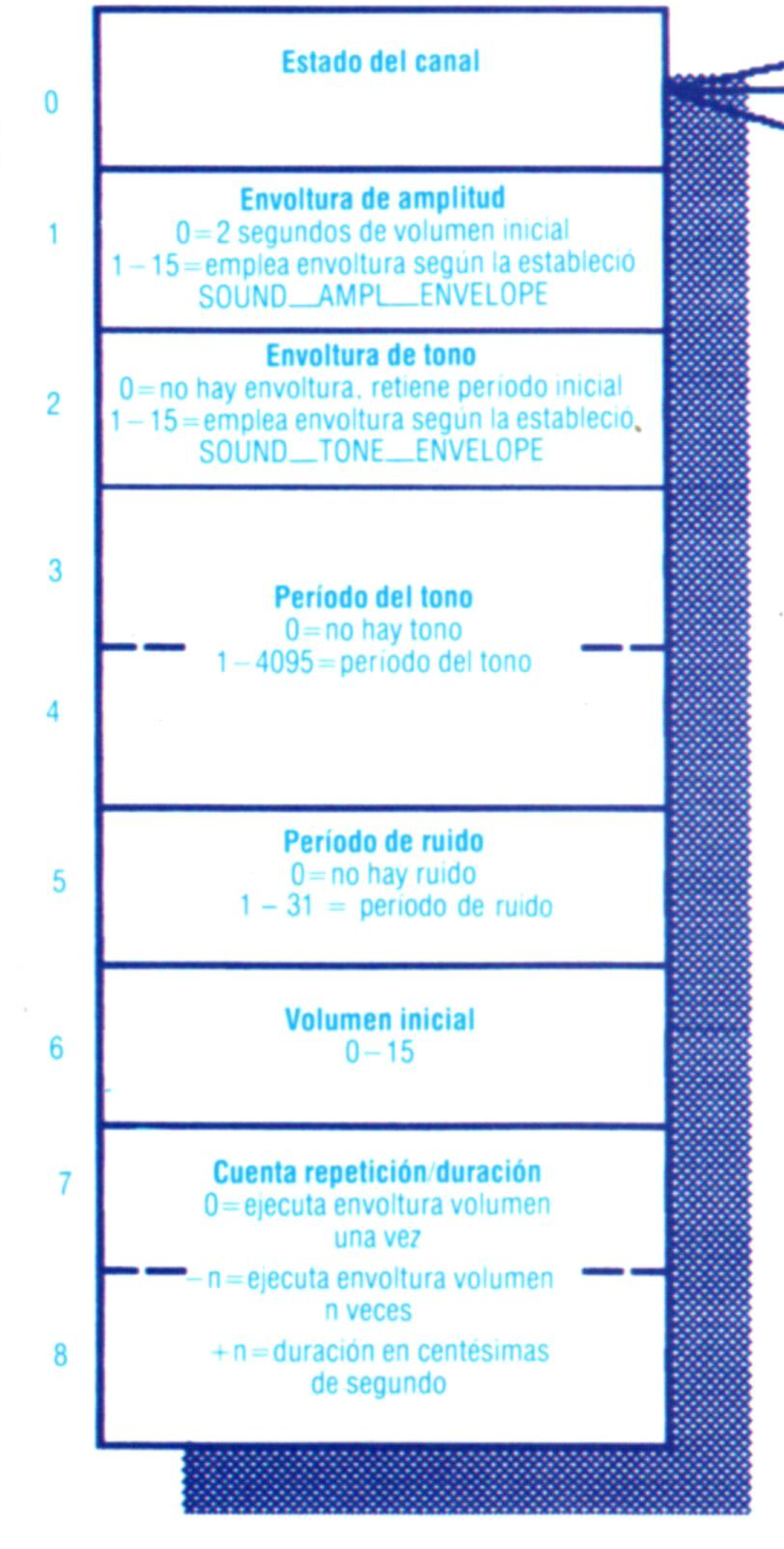
El 8912 soporta tres canales de sonido independientes (A,B,C) y un generador seudoaleatorio de sonido, que puede programarse para que aparezca en cualquier canal. El chip ofrece las facilidades básicas para producir un sonido y para cambiar el volumen del sonido, mediante envolturas de sonido predefinidas. Las salidas del generador de sonidos se mezclan para producir una salida stereo: el canal A es el de la izquierda, el canal B el de la derecha, y el canal C es una mezcla equitativa de los dos. Estas facilidades han sido ampliadas por el sistema operativo, que proporciona el control software del sonido y del volumen a un tiempo con métodos de sincronización de canales sonoros.

Las instrucciones BASIC tales como ENV, ENT, RELEASE, SOUND y SQ se sirven directamente del sistema operativo para obtener sonidos. De hecho los parámetros que se pasan y los conceptos correspondientes son casi idénticos. El sistema operativo controla el sonido por medio de un evento de interrupción especial que se produce un centenar de veces por segundo. Este evento está destinado a procesar las instrucciones de sonido; así, por ejemplo, la cita de los canales y la actualización de las envolturas tienen lugar durante este evento. Consideremos primero las instrucciones de sonido del Amstrad y el uso de las colas al procesarlas.

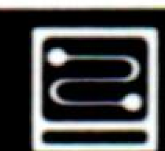
Todo sonido que deba generarse tiene asignado un bloque de instrucciones que contiene parámetros destinados a especificar la nota y el volumen iniciales, las envolturas de volumen y de tono que siguen y los detalles de sincronización y duración. El bloque de instrucciones de sonidos, que debe encontrarse en los 32 K centrales de memoria, puede colocarse en una cola de sonido del canal con la rutina SOUND_QUEUE. Los parámetros que se pasan tienen el mismo significado que los empleados en la instrucción SOUND del BASIC, aunque especificados en orden diferente. El esquema (izquierda) ilustra los detalles de la estructura de las instrucciones de sonidos que se necesitan para SOUND_QUEUE. Cuando la instrucción de sonido alcanza la parte superior de la cola se ejecuta si se cumplen los requisitos de sincronización. De otro modo puede esperar su liberación por medio de SOUND_RELEASE o un encuentro con otro canal de sonido.

La rutina SOUND_CHECK toma un número de bit de canal del acumulador, y devuelve al mismo el estado del canal, con las mismas asignaciones que la función BASIC SQ().

Examinemos ahora el control de las envolturas. Una vez más, las entradas del firmware para establecer las envolturas de volumen y tono toman los mismos parámetros que las instrucciones equivalentes BASIC. Los datos para las envolturas se alma-



Bloque de instrucciones sonoras
Una instrucción de sonido consiste en un bloque de datos que contiene varios parámetros, semejantes a los que se usan en la instrucción SOUND del BASIC. La dirección de este bloque se pasa en HL hasta SOUND_QUEUE, que después realiza la oportuna operación. Si la cola está llena, la instrucción se ignora y el flag de arrastre se hace falso



cenan como una tabla en la forma que se pasan a las rutinas, y posteriormente se procesan cada vez que es necesaria la envoltura. La dirección del bloque de datos asociado con una envoltura puede determinarse en cualquier momento por medio de SOUND_T_ADDRESS, o con SOUND_A_ADDRESS, por lo que es posible alterar una envoltura sin llamar al firmware parcheando directamente el bloque de datos.

El formato del bloque de datos es el que mostramos en el diagrama de formato de datos de envoltura (derecha); se puede parchear cualquier sección con sólo calcular el desplazamiento necesario en el bloque de datos. Si se adopta este método, hay que tener cuidado, ya que el firmware puede acceder simultáneamente al área de datos mientras es parcheada. Aunque no se van a producir grandes desastres, es muy posible que el sonido producido no sea como se ha previsto.

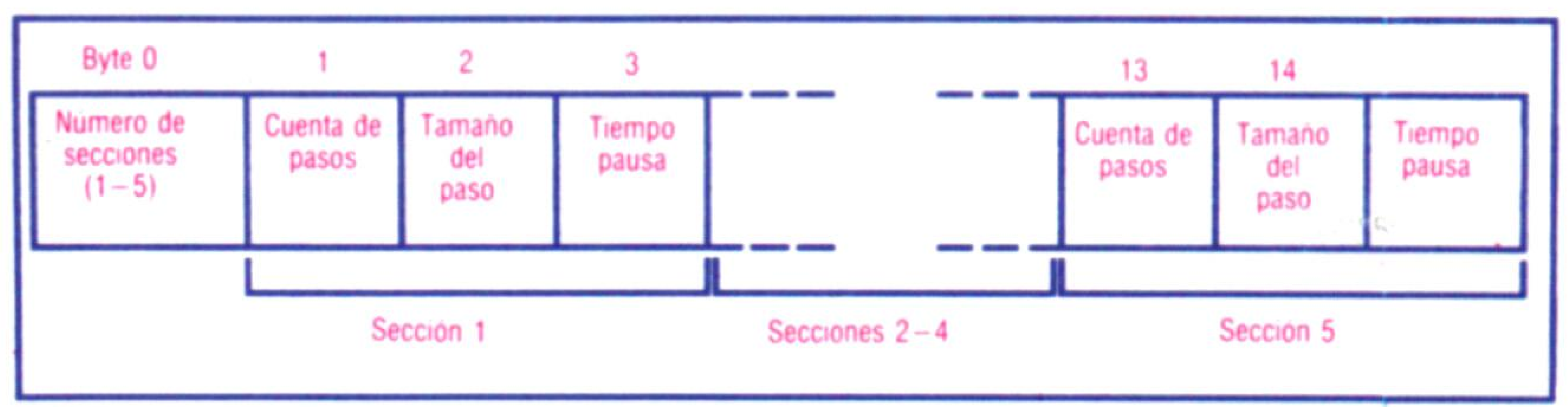
El firmware proporciona una rutina especialmente útil, que es la SOUND_ARM_EVENT. Ésta establece un evento que ha de lanzarse cuando la cola de los sonidos para un siguiente canal especificado se encuentra vacía. De hecho, la rutina lleva un evento especial a la interrupción que procesa el sonido, que sucede cada centésima de segundo. Este evento comprueba la cola de sonidos del canal en cuestión y, si hay espacio, entonces es lanzado el bloque de eventos pasado a la rutina SOUND_ARM_EVENT. Este evento lanzado se procesa después en la manera especificada en su clase de evento.

El bloque de eventos que se pasa debe ser inicializado primero empleando KL_INIT_EVENT, por lo que puede tener cualquier clase o prioridad. Sin embargo, dado que la interrupción de sonido sucede cada centésima de segundo, es aconsejable establecer el evento según el tipo asíncrono normal.

Este evento puede ser usado para generar música continua de fondo a un programa sin que el programador tenga que preocuparse de proporcionar continuamente datos al gestor del sonido. En su lugar, la rutina de eventos debe leer la siguiente dirección del programa del sonido desde el área predefinida de datos y después llamar la entrada SOUND_QUEUE. Se puede disponer la rutina de tal modo que vuelva circularmente al inicio de los datos cuando llegue al extremo final con el fin de generar un efecto musical continuo.

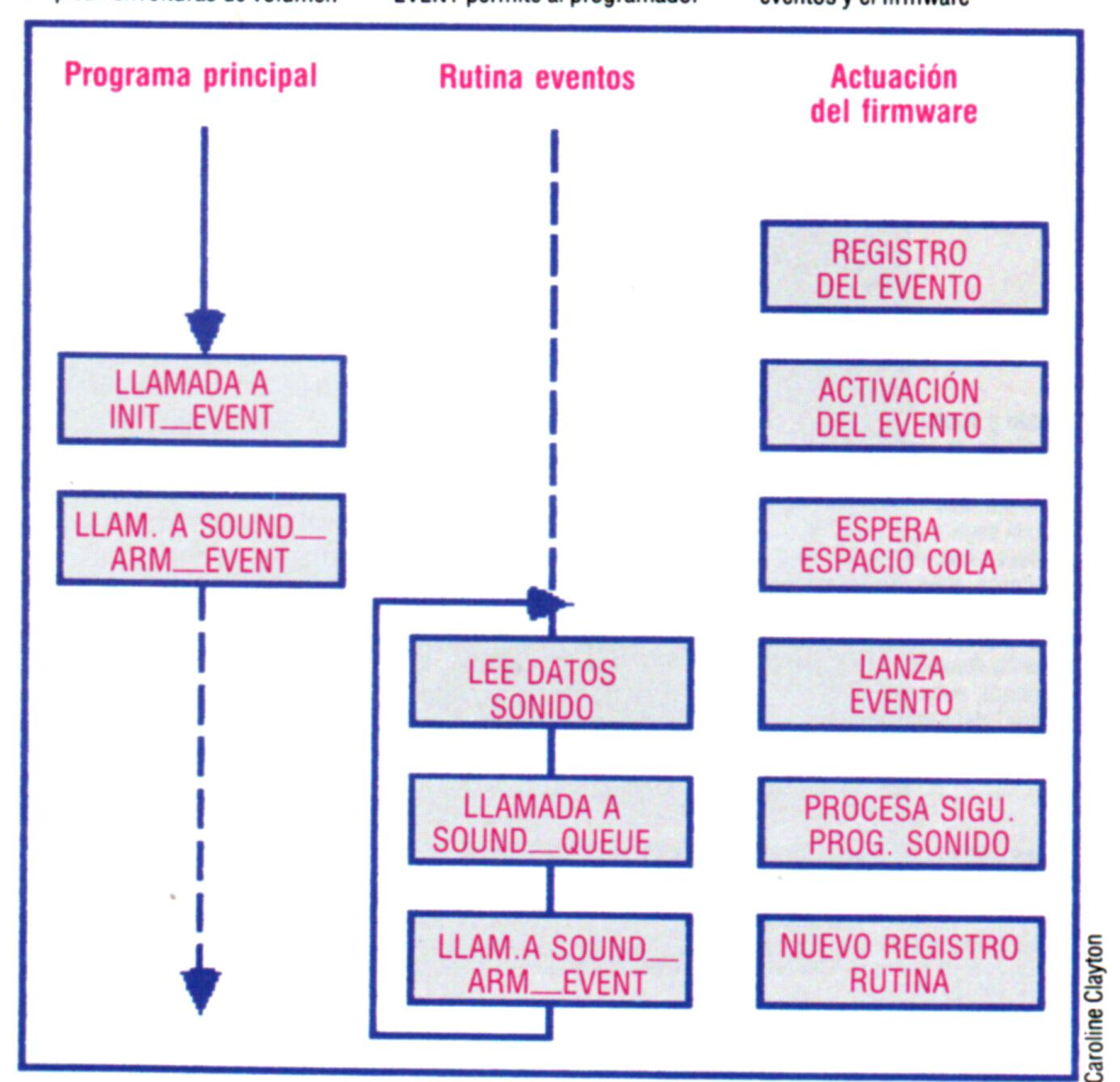
La entrada SOUND_QUEUE desactiva el evento de sonido, de modo que antes que concluya la rutina debe registrarse llamando a SOUND_ARM_EVENT, que pasa su propia dirección del bloque de eventos. El diagrama de flujo muestra cómo interaccionan la rutina de eventos y la rutina de cola de sonidos para generar el efecto musical continuo.

La generación de sonidos puede detenerse en cualquier momento mediante la entrada SOUND_HOLD; ésta suspende toda envoltura actualmente en uso y desactiva el evento de sonido (si está activado). El sonido puede volverse a iniciar llamando SOUND_CONTINUE, SOUND_QUEUE o bien SOUND_RELEASE. De estas rutinas la última se da para iniciar cualquier programa de sonido que esté señalizado (*flag*) para ser retenido individualmente. Se llama SOUND_HOLD siempre que se detecta una parada (*break*) desde el teclado, para asegurarse de que no se permite continuar ninguna generación de sonidos espuria desde dentro de un programa.



▲ **Bloque de datos de envoltura**
Toda envoltura puede tener hasta cinco secciones, cada una de las cuales se especifica mediante tres bytes que determinan la cuenta de los pasos, el tamaño de los pasos y el tiempo de pausa. Si se emplean envolturas de volumen del hardware, el primer byte del bloque retiene el número de la envoltura (entre el 128 y el 143) y los bytes dos y tres retienen el tiempo de pausa

▼ **Evento notable**
El empleo de SOUND_ARM_EVENT permite al programador generar música controlada por eventos en los Amstrad CPC que funcionará como fondo mientras continúa ejecutándose el programa principal en primer plano. El esquema muestra las operaciones efectuadas por el programa principal, la rutina de eventos y el firmware

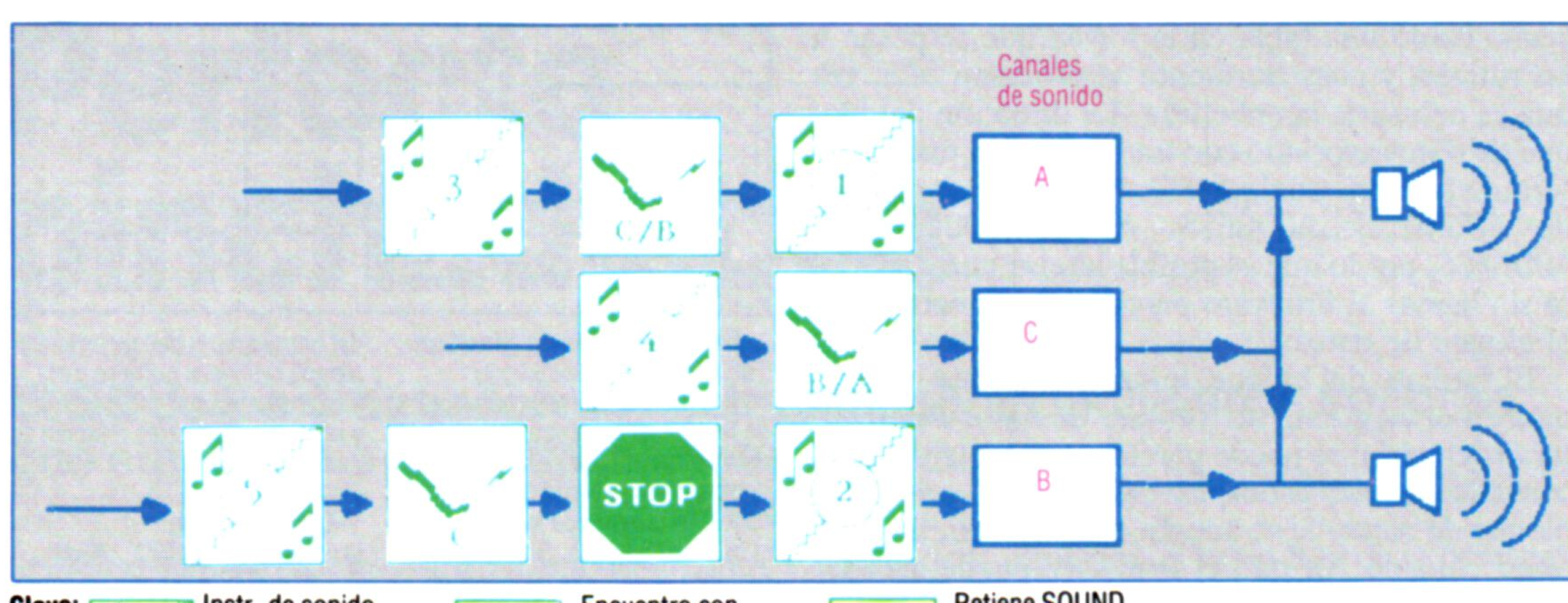


Entradas del gestor de sonidos

OBCA7H	SOUND_RESET	Restablece gestor sonidos
OBCAAH	SOUND_QUEUE	Añade un sonido a una de las colas
OBCADH	SOUND_CHECK	Da el estado de una cola de sonido
OBCB0H	SOUND_ARM_EVENT	Permite el lanzamiento de un evento cuando la cola de sonido está vacía
OBCB3H	SOUND_RELEASE	Libera los sonidos contenidos en una cola
OBCB6H	SOUND_HOLD	Detiene todo sonido en todas las colas
OBCB9H	SOUND_CONTINUE	Reanuda los sonidos detenidos
OBCBCH	SOUND_AMPL_ENVELOPE	Inicializa una de las envolturas de volumen del software
OBCBFH	SOUND_TONE_ENVELOPE	Inicializa una de las envolturas de tono del software
OBCC2H	SOUND_A_ADDRESS	Da la dir. de una envolt. de volumen
OBCC5H	SOUND_T_ADDRESS	Da la dirección de una envolt. de tono

Los detalles completos de las direcciones de entrada y salida de estas rutinas se encuentran en la publicación de Amsoft *SOFT 158*, el manual de especificaciones del firmware del Amstrad

Encuentro con el canal
En la situación que aquí se representa se están ejecutando los sonidos 1 y 2. Cuando 2 concluye, la instrucción siguiente en la cola para el canal B se retiene a la espera de SOUND_RELEASE. Puesto que la instrucción en la cola para C requiere un encuentro con los canales A y B, es retenida a la espera de las últimas instrucciones sobre estas colas. Mientras tanto, 1 termina de sonar y el canal A se encuentra con C, pero todavía espera su encuentro con B. Tan pronto como la cola para B es liberada se hace efectivo el encuentro, y se oirán simultáneamente los sonidos 3, 4 y 5



Clave: Instr. de sonido (número indicado)


Encuentro con canal indicado



Retiene SOUND_RELEASE hasta ser liberada

Sonido y visión
La edición de envolturas de sonido puede ser un asunto complejo, sobre todo si usted no está seguro de cómo se produce exactamente el efecto que desea. Nuestro *Editor de envolturas* le permite especificar parámetros a su elección, y visualiza después la forma de la envoltura, mostrándole la "figura" del sonido que ha creado

Una llamada BASIC

La instrucción CALL del BASIC no está del todo documentada en los manuales del Amstrad. Permite pasar parámetros a rutinas en código máquina en un "bloque de parámetros". Los parámetros que da el usuario se almacenan por orden; el primer parámetro se retiene en la dirección del bloque más baja (la primera). Esta dirección viene apuntada por IX a la entrada a la rutina llamada (CALL). Los valores numéricos pueden ser pasados directamente, mientras no pueden serlo las variables. En contrapartida, la dirección de la variable (o, en el caso de una variable de cadena, la dirección del bloque descriptor de la cadena) es pasada empleando el símbolo @. Así, p.ej., si deseamos llamar (CALL) una rutina en &A000, pasándole el valor 85 y volviendo al BASIC con un resultado almacenado en las variables result% y answer$, escribiríamos:

```
CALL &A000,85,@ result%,@ answer$
```

Al entrar en la rutina, IX apuntará a un bloque en la RAM que contiene primero el valor 85 almacenado en forma de entero sin signo de 16 bits, después dos bytes que contienen la dirección de result% y finalmente la dirección del bloque descriptor de la variable en cadena para answer$. El bloque descriptor de la cadena comprende tres bytes que contienen el valor de la longitud de la cadena, y su dirección. Las rutinas del usuario deberán evitar la alteración del contenido de este bloque y el procesamiento de cadenas habrá de efectuarse con cuidado para no tener problemas con el firmware.

Editor de envolturas

```
50 MEMORY &6FFF: DIM envelope(15)
60 done=0: routine=&7000: buffer=routine+&100: nk=1
70 WHILE -1
80 MODE 2: LOCATE 1,20: INPUT "Inicio volumen (0-
   15)";sv%:I F sv%>15 OR sv%<0 THEN 80
90 LOCATE 1,21:INPUT "Inicio tono";tt:,IF tt<0 OR tt>4095
   THEN 90
100 LOCATE 1,22:INPUT "Escala horizontal (1-10)";sc%: IF
    sc%>10 OR sc%<1 THEN 100
110 LOCATE 1,23:INPUT "Envoltura volumen o tono
    (V/T)";senv$:IF (UPPER$(senv$)<>"V")AND (UPPER$)
    (senv$)<> "T") THEN GOTO 110
120 W=0: IF UPPER$(senv$)="V" THEN pk=&BC: W=1:
    ELSE pk=& BF
130 length%=0: soundx=0: IF W=1 THEN
    soundy=INT(sv%*9.7) ELSE soundy=68
140 FOR x=1 TO 15
150   envelope(x)=o
160 NEXT
170 GOSUB 270
180 LOCATE 1,20: PRINT CHR$(18)
190 FOR count=1 TO 5
200 GOSUB 360: nk=ABS(nk-1): GOSUB 540: IF W=1 THEN
    GOSUB 880 ELSE GOSUB 890
210 NEXT
220 WHILE NOT done
220 LOCATE 5,20: INPUT "Edit (y/n)";ed$
240 IF UPPER$(ed$)="N" THEN done=-1 ELSE GOSUB 790
250 WEND
260 WEND
270 REM inicializa pantalla
280 MODE 1:WINDOW # 1,1,39,1,2: GOSUB 1000
290 MOVE 20,380: DRAW 20,210,3: DRAW 600,210,3
300 MOVE 21,soundy+211
310 POKE routine, &3E: POKE routine+1,1
320 POKE routine+2,&21: POKE routine+3, buffer-
    (INT(buffer/256))*256: POKE routine+4, INT(buffer/256)
330 POKE routine+5,&C3: POKE routine+6,pk: POKE
    routine+7,&BC
340 POKE routine+8,&C9
350 RETURN
360 REM toma datos sección
370 LOCATE 1,20: PRINT CHR$(18);: LOCATE 5,20: PRINT
    "Sección";count
380 LOCATE 1,22: PRINT CHR$(18);: LOCATE 5,22: INPUT
    "Contador pasos"; skip%: IF W=0 THEN GOTO 400 ELSE IF
    skip%<0 OR skip%>127 THEN 380
390 GOTO 410
400 IF skip%<0 OR skip%>239 THEN GOTO 380
410 P$=STR$(SKIP%): GOSUB 900
420 LOCATE 1,23: PRINT CHR$(18);: LOCATE 5,23: INPUT
    "Tamaño pasos";size%: IF W=0 THEN GOTO 440 ELSE IF
    ABS(size%)>15 THEN 420
430 GOTO 450
440 IF ABS(size%)>127 THEN 420
450 P$=STR$(size%): GOSUB 900
460 ss=size%: IF size%<0 THEN size%=256-ABS(size%)
470 LOCATE 1,24: PRINT CHR$(18);: LOCATE 5,24: INPUT
    "Tiempo pausa"; pause%: IFpause%>255 THEN 470
480 P$=STR$(pause%): GOSUB 900
490 envelope (<count-1)*3+1)=skip%: envelope
    ((count-1)*3+2)=size%: envelope
    ((count-1)*3+3=pause%
500 FOR x=20 TO 24
510   LOCATE 1,x: PRINT CHR$(18);
520 NEXT
530 RETURN
540 REM inicialización envoltura
550 POKE fuffer,count
560 CLS #1: GOSUB 1000
570 FOR x=0 TO count-1
580 POKE buffer+(x*3+1),envelope(x*3+1):
    skip%=envelope(x*3+1)
590 p$=STR$(envelope(x*3+1)): GOSUB 900
600 POKE buffer+(x*3+2),envelope(x*3+2):
    size%=envelope(x*3+2)
610 pp=envelope(x*3+2): IF pp>127 THEN pp=pp-256
620 p$=STR$ (pp): GOSUB 900
630 POKE buffer+(x*3+3),envelope(x*3+3):
    pause%=envelope(x*3+3)
640 p$=STR$(envelope(x*3+3)): GOSUB 900
650 NEXT x
660 length%=length%+skip%*pause%
670 CALL routine
680 IF W=0 THEN GOSUB 910: RETURN
690 off%=size%*10: IF size%>127 THEN
    off%=(size%-256)*10
700 FOR x=1 TO skip%
710   soundy=(soundy+off%) MOD 150: IF soundy<0 THEN
      soundy=150-soundy
720 FOR c=1 TO (10/51)*pause%+2: soundx=soundx
    + (sc%)
730   DRAW soundx+21,soundy+211,2+nk
740   NEXT
750 NEXT
760 DRAW soundx+21,soundy+211,2+nk
770 RETURN
780 REM dibuja gráfico otra vez
790 LOCATE 5,20: PRINT CHR$(18);: INPUT "Seccion";sec%:
    IF sec%<1 OR sec%>5 THEN 790
800 count=sec%: GOSUB 360
810 CLS: soundx=0: soundy=INT(sv%*9.7): length%=0:
    GOSUB 270
820 FOR section=1 TO 5
830   count=section: GOSUB 540
840 NEXT
850 GOSUB 870
860 RETURN
870 REM sound
880 SOUND 1,tt,length%,sv%,1: RETURN
890 SOUND 1,tt,length%,sv%,,1: RETURN
900 PRINT #1,MID$(P$,(2-ABS(VAL(p$)<0)));",";:RETURN
910 FOR x=1 TO skip%
920    soundy=soundy-((15/127)*ss): IF soundy<0 THEN
    soundy=0
930 IF soundy>189 THEN soundy=189
940 FOR c=1 TO (7/51)*pause%+2: soundx=soundx+sc%
950   DRAW soundx+21,soundy+211,2+nk
960   NEXT
970 NEXT
980 DRAW soundx+21,soundy+211,2+nk
990 RETURN
1000 IF W=1 THEN PRINT #1, "ENV 1, "; ELSE PRINT #1,
     "ENT 1 , ";
1010 RETURN
```


mi computer
Editorial Delta, S.A.



10106
9 788485 822836